Proceso ETL Masivo de XMLs

Por
Mario Heredia

Tutora:
Ing. Daniela Díaz

Facultad:
Facultad de Ciencias de la Administración

Carrera:
Ingeniería en Sistemas

Modalidad:
Proyecto de Grado

Cordoba, Argentina. Noviembre 2016.

Declaración de derechos de autor

Agradecimientos

A Dios por permitirme hacer lo que siempre me gusto y que a pesar de mis errores, el me siga amando y dando el ingenio y la creatividad para construir soluciones como la del presente trabajo.

Muchas gracias Papá.
Tu hijo.

Dedicatoria

A Santiago Heredia y Luciana Martínez, por ayudarme a culminar esta etapa de mi vida dándome de su tiempo para concretar este sueño.
Y como son y siguen siendo parte de mi vida, este trabajo se los dedico a ustedes.

Resumen

Dentro de la jefatura de "Performance de Red" una de las principales fuentes de información sobre el estado de la red de telecomunicaciones es provista por Nokia.

Actualmente Nokia se encuentra implementando nuevos features a sus equipos y con el objeto de ofrecer un mejor servicio, se implementaron cambios en la forma en que se entregan los datos de la performance de la red de telecomunicaciones, dando origen al actual trabajo.
El principal cambió de Nokia fue cerrar el acceso a sus BD y otorgar permiso de lectura a sus servidores, permitiendo obtener la información ahora en un formato distinto, en archivos XML y no más desde las tablas de la BD.

El presente trabajo tiene como problema central construir una solución que permita, a la jefatura, continuar recibiendo y procesando en tiempo y en forma la información de la performance de la red de telecomunicaciones, publicándola como lo venía haciendo y de manera transparente al resto de la compañía.

Se implementara una metodología ágil para la resolución del problema y para el seguimiento del proyecto. En el momento inicial del proyecto, en donde estábamos definiendo el alcance y el cómo lo íbamos a llevar a cabo, nos pareció correcto manejarlo así e ir avanzando incluso por etapas para tener una devolución mas rápida y corregir los errores o para implementar soluciones de ser necesario.

Se decide avanzar entonces por etapas, cerrándolas con alguna funcionalidad finalizada, garantizando que funciona y que no era necesario volver a revisar ese punto o proceso de nuevo.

La solución entonces es un proyecto conformado por cuatro o cinco etapas, según como se mire, que culminan con el componente finalizado. Al final de la última etapa, se integran los componentes y se da por concluido el proyecto. Luego de algunas pruebas integrando todos los componentes, se dirá que la solución es productiva y está lista para comenzar a funcionar.

Índice de Contenido

Índice de Contenido 6
Índice de Tablas 8
Índice de Graficas 8
Introducción 10
Situación Actual 10
Situación Problemática 13
Problema 13
Objeto de estudio 13
Campo de Acción 13
Objetivos 13
Objetivo General 13
Objetivos Específicos 14
Idea a defender | Propuesta a Justificar | Soluciones a comprobar 14
Delimitación de proyecto 14
Aporte Práctico 15
Aporte Teórico 15
Método de investigación 16
Primera Parte. Marco Contextual. 17
Entorno del objeto de estudio 17
Relación tesista y objeto de estudio 18
Análisis de los problemas observados 19
Presentación de los datos 19
Diversidad 19
Frecuencia 19
Escalabilidad 19
Antecedentes de proyectos similares 20
Segunda Parte. Marco Teórico. 21
Marco Teórico del Objeto de Estudio 21
Open Measurement Standard (OMeS) 21
Convención del nombre del archivo OMeS 22
Marco teórico del campo de acción 22
Diagnóstico 34
Tercera Parte. Modelo Teórico. 35
Planificación 35

Requerimientos 38
Diagrama de contexto 38
Elemento External Storage 38
Elemento Internal Storage 39
Componente de Transposición 39
Componente de Estandarización 39
Componente Unificador. 40
Análisis y Diseño 41
Diagramas de descomposición funcional de los subprocesos existentes 41
Diagrama del Proceso de Conexión a Servidor Remoto 41
Diagrama de identificación de archivos 42
Diagrama de transposición de valores. 42
Diagrama de Homologación/Estandarización de valores 43
Diagrama de unificación de valores 44
Diagramas de Clases 45
Riesgos del proyecto 46
Escenario desordenado 47
Escenario incompleto 48
Escenario con excedente de datos 48
Escenario complejo 49
Cuarta Parte. Concreción del modelo. 50
Implementación 50
Arquitectura del Proceso 52
Estructuración de los componentes 52
Paths 52
Storage 52
Componentes y Funciones 53
Puesta en Marcha 53
Conclusiones 53
Bibliografía 55
Glosario 56
Anexo 58
Construcción del escenario de confianza para las conexiones SSH sin contraseña 58
Estructura del proceso 66
Guía Troubleshooting. 67

Índice de Tablas

Tabla 1.1.1. Relaciones entre las distintas entidades objeto..... 18
Tabla 2.1.1. Componentes del estándar OMeS..... 21
Tabla 2.1.2. Convención del esquema de nombres para los OMeS files..... 22
Tabla 3.1.1 Detalle Macro del proyecto, etapas y tiempos..... 35
Tabla 3.1.2 Detalle a nivel de actividad del proyecto, etapas y tiempos..... 36
Tabla 3.3.1. Ejemplo de situación con valores desordenados..... 47
Tabla 3.3.2. Ejemplo de situación con valores incompletos..... 48
Tabla 3.3.3. Ejemplo de situación con valores de más..... 48
Tabla 3.3.4. Ejemplo de situación de múltiples escenarios con valores desordenados, incompletos y de más..... 49
Tabla 4.1.1. Esquema de directorios..... 52
Tabla 4.1.2. Relación de directorios, regional por regional, con los archivos XML dentro del Regional Cluster y su ubicación destino dentro del Storage de la jefatura..... 52
Tabla 4.1.3. Orden de ejecución, nombre de los componentes de la solución y su ubicación dentro del esquema de directorios..... 53

Índice de Graficas

Arquitectura de Red GSM..... 11
Arquitectura de Red UMTS..... 11
Arquitectura de Red LTE..... 12
Arquitectura en Conjunto. GSM, UMTS y LTE..... 12
Grafico 2.1.1. Contenido de un XML según convención OMeS..... 22
Grafica 2.2.1. Diagrama de Flujo de como procesa las sentencias AWK..... 24
Grafica 3.1.1 Diagrama de Gantt de las actividades del proyecto..... 37
Grafica 3.2.1. Diagrama de Contexto de la Solución Parser..... 39
Grafica 3.3.1. Diagrama de Descomposición Funcional. Proceso de Conexión y de Recolección de archivos..... 41
Grafica 3.3.2 Diagrama de descomposición funcional. Proceso de Identificación de archivos..... 42
Grafica 3.3.3. Diagrama de descomposición funcional. Proceso de Transposición de valores..... 42
Grafica 3.3.4. Diagrama de descomposición funcional. Proceso de Homologación de valores..... 43
Grafica 3.3.5. Diagrama de descomposición funcional. Proceso de Unificación de Valores. 44
Grafica 3.3.6. Diagrama de Clases del proyecto..... 45
Grafica 3.3.7. Diagrama de actividades. Proceso de establecimiento de conexión sin contraseña..... 46
Grafica 4.1.1. Construcción de la variable folder..... 50
Grafica 4.1.2. Salida de la función f_getDateHour_yyyyMMddHH..... 50
Grafica 4.1.3. Sentencias de invocación de funciones Unix y programas AWK..... 50
Grafica 4.1.4. Bucle del tipo FOR, con sentencias que se ejecutaran por cada regional..... 51

Grafica A.1.1 Generación de Claves Asimétricas. La Pública y la Privada..... 59
Grafica A.1.2. Visualización correcta de claves en .ssh folder..... 60
Grafica A.1.3. En el Servidor, creando el archivo authorized_keys..... 61
Grafica A.1.4. Copiando la clave pública en el Servidor..... 62
Grafica A.1.5. Ingresando la clave pública del cliente dentro del archivo de claves autorizadas (authorized_keys)..... 63
Grafica A.1.6. Comprobación del éxito en la creación del escenario de confianza..... 64
Grafica A.2.1. Esquema general de la solución..... 65
Grafica A.3.1. Diagrama de Flujo de esquema troubleshooting..... 67

Introducción

En la empresa Claro AMX, existe la jefatura de "Calidad y Performance de Red" y se encuentra dentro de la gerencia de "Calidad de Red", a su vez también, dentro de la dirección de "Ingeniería".
Una de las principales tareas de la jefatura de Performance es proveer información del estado de la red de telecomunicaciones para análisis y toma de decisiones a diferentes jefaturas dentro de la gerencia de Calidad e incluso también a otras gerencias.
Y para cumplir esa tarea la jefatura cuenta con dos grupos, un grupo de analistas y un grupo de desarrolladores.
Los desarrolladores son los responsables del análisis, diseño, desarrollo, prueba e implementación de procesos orientados a la recolección, transformación y almacenamiento de datos, los cuales, todos o la gran mayoría provienen de distintos orígenes, en diferentes formatos y en distintos periodos de tiempo, que en conjunto forman la fuente de información que nutre la BD para la construcción de nueva información.
Los analistas, en cambio, son los responsables de la construcción de reportes y de análisis del contenido, marcan tendencias en la red como también sugieren acciones correctivas o preventivas según corresponda.

Situación Actual

Una de las fuentes de datos que cuenta la jefatura para el soporte a la toma de decisiones, es proporcionada por la empresa Nokia Siemens Networks (NSN de ahora en adelante) y contiene datos hora por hora del funcionamiento de cada una de las celdas de la red.
Lo cierto es que NSN presento nuevos features en sus equipos y sus BDs no fueron la excepción. A medida que la red de telecomunicaciones de Claro iba creciendo, NSN acompañó su crecimiento ampliando su capacidad de hardware y el tamaño de sus BDs, proporcionando toda clase de información sobre el desempeño de las celdas de la red, los cuales están a disposición de la jefatura.
Esa disposición nunca cambió los datos siempre estuvieron disponibles, lo que cambió fue el "como" estaban disponibles.

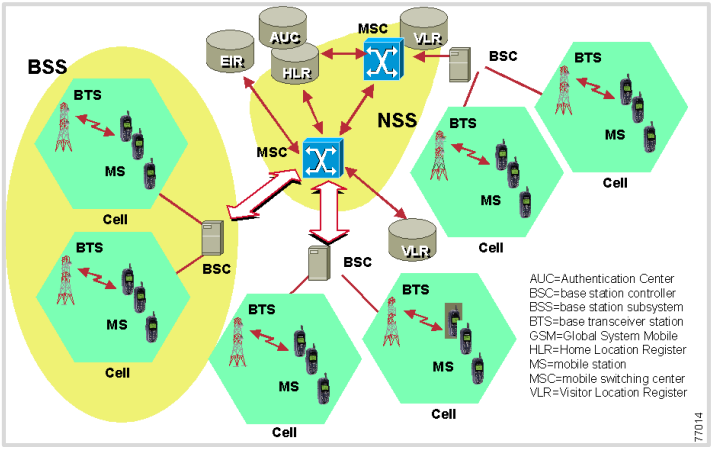
El acceso a los datos de desempeño de cada uno de los elementos de la red de telecomunicaciones de Claro, es el motivo del proyecto, aquí es donde se centra el problema y se construye la solución.

La jefatura de Calidad de Red extraía los datos directamente de las BDs de NSN, mediante dbLinks se tomaban datos directamente de cada una de las BDs de NSN y se almacenaban en la BD de la jefatura para su posterior tratamiento y análisis.

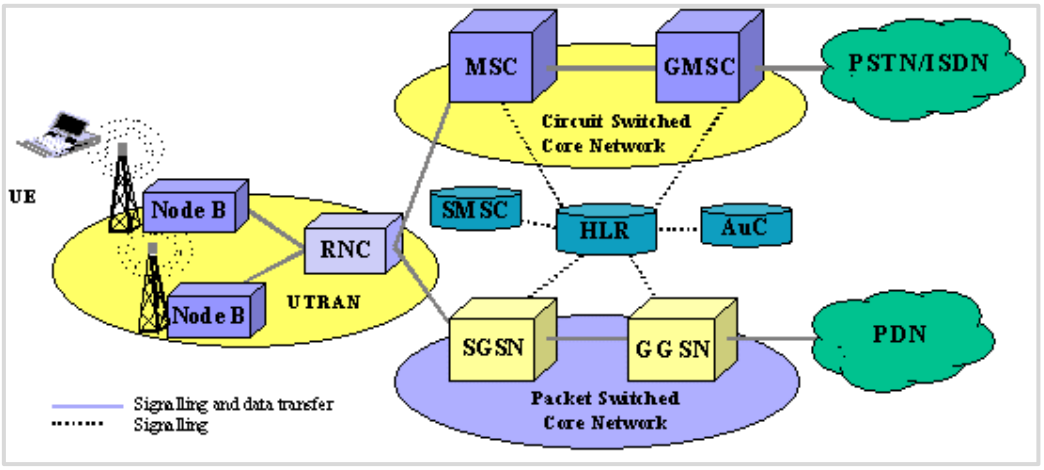
Cuando NSN propuso a Claro la implementación de un pack que contenía nuevas funcionalidades en herramientas de monitoreo, nuevos datos de desempeño de las celdas de la red y una nueva estructura virtual de su actual hardware para una gestión más ágil y escalable, junto con este pack, también venia la mejora de cerrar todos los accesos directos a sus BDs y en su reemplazo proponía la creación de un repositorio de archivos XML con los datos de desempeño de las celdas de la red.

El proyecto en cuestión se centra en este nuevo problema: Cambiar la forma en que se obtienen los datos, para ajustarse al nuevo escenario.

Red GSM. Esquema de la arquitectura.

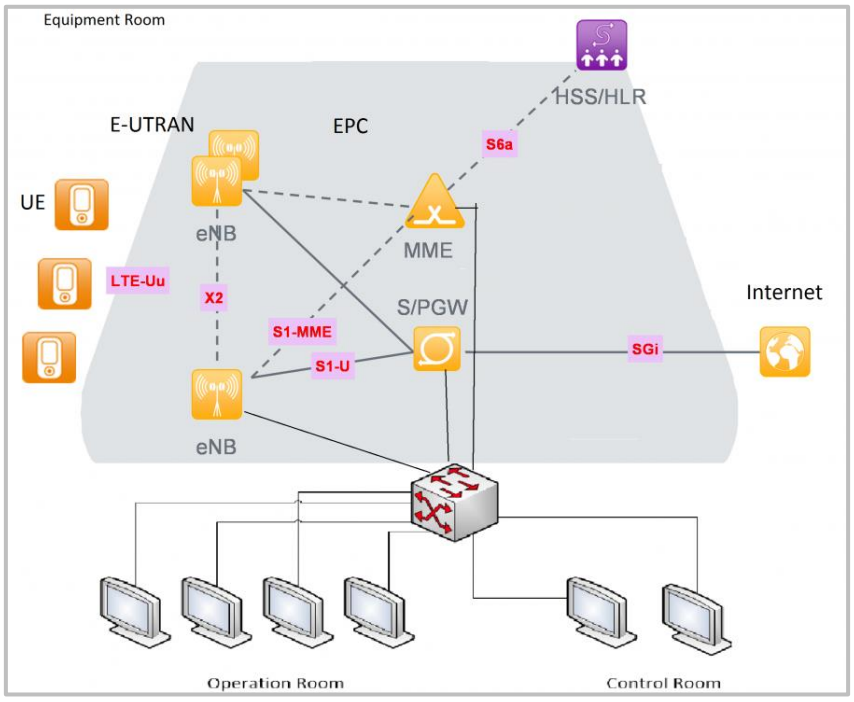


Red UMTS. Esquema de la arquitectura.



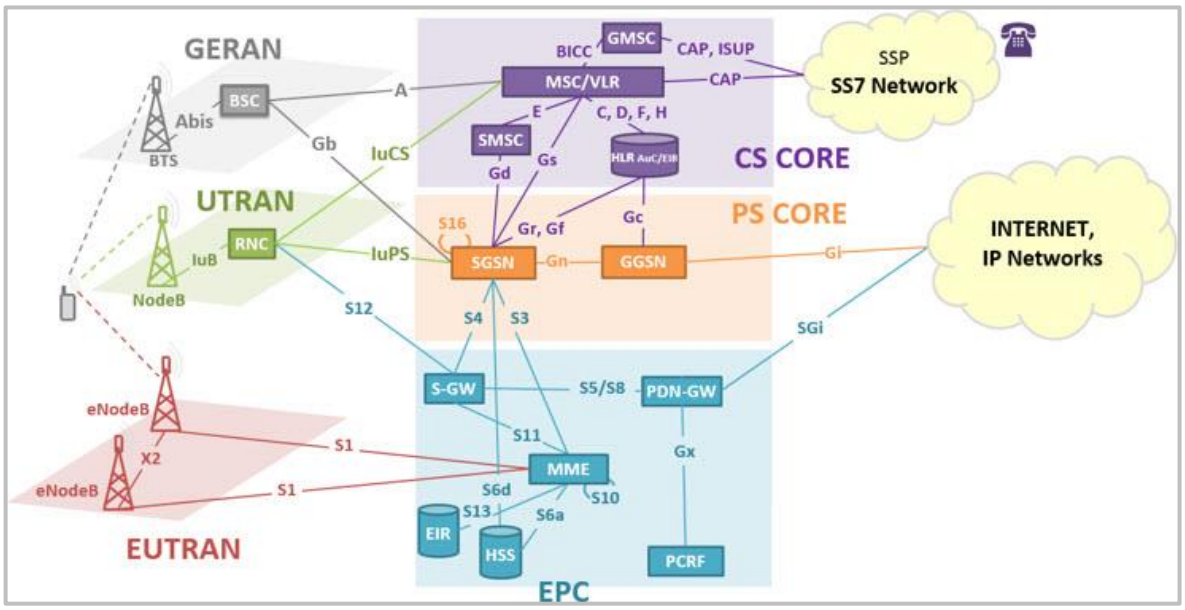
El NodeB y el RNC son los elementos que generan las mediciones de nuestro interés. Conforman lo que se denomina sector UTRAN en la gráfica. Dentro de las mediciones se encuentran las interfaces Iub (comunicación entre NodeB e RNC) e Iur (comunicación entre RNCs).
El NodeB contiene las celdas que el elemento controlador (RNC) se encarga de controlar y monitorear.

Red LTE. Esquema de la arquitectura.



LTE es la tecnología más joven y por ende en etapa de crecimiento, que sus elementos (como la gráfica los indica) sean pocos no significa que sea menos compleja.

Redes todas en Conjunto. Un esquema de la realidad.



Aquí se puede ver las redes en conjunto, las diferentes tecnologías y los elementos de cada una. Esta grafica es a modo informativo.

Situación Problemática

Los desafíos son muchos y muy variados, además muchos factores entran en juego pero el factor tiempo es el principal. Hasta este momento las fuentes de información externas que se manejan (es decir, dentro de la jefatura) son: archivos con datos en columnas (.txt), archivos con valores separados por coma (.csv) y en algunos casos la salida de la ejecución de comandos (.log).
Pero nunca se han trabajado con archivos de tipo de XML, no solamente que nunca se ha trabajado con esta clase de archivos sino que además, las herramientas que se encuentran en el mercado proporcionan soluciones muy complejas a nivel de código, es decir, mucho código y poco claro.
Las primeras pruebas de concepto (POC) arrojaron resultados muy desalentadores.
No era factible procesar de manera dinámica ágil, sencilla y eficiente el volumen de archivos XML que este nuevo escenario nos proporcionaba, los tiempos comenzaron a acortarse y es en este mismo contexto donde se toma la decisión de comenzar a construir un proceso "parseador" propio, nuestro, y que cubra todas esas exigencias que necesitamos.

Problema

Las soluciones de proveedores, no pueden procesar de manera dinámica, ágil, sencilla y eficiente el volumen de archivos XML que este nuevo escenario nos proporciona y es en este contexto donde se toma la decisión de comenzar a construir un proceso "parseador" propio, nuestro, y que cubra todas esas exigencias que necesitamos.

Objeto de estudio

El objeto de estudio es sin lugar a dudas el proceso "parseador", es decir, todas las etapas que tienen como punto de partida el archivo XML y que concluyen en un archivo con un formato estándar fácil de procesar.
El foco de trabajo es precisamente ese, el transformar lo complejo de un XML, de una manera rápida y sencilla en un archivo con formato estándar.

Campo de Acción

El entorno de trabajo está delimitado por el tipo de sistema operativo y por el lenguaje de programación, los cuáles son Unix y AWK respectivamente.
Con respecto al lenguaje de programación, AWK se invocaría desde código Shell de Unix.

Objetivos

Objetivo General

- El objetivo del proyecto es armar una solución escalable y funcional que reemplace el método de recolección de datos actual, permitiendo parsear una gran cantidad de XML, que de ser necesario se pueda agregar nuevas mediciones y nuevos elementos para ser procesados, persistiendo luego toda la información en una base de datos y todo en un tiempo menor a una hora, de esta manera, la jefatura continua proveyendo la información de la performance de la red a todas las áreas de la compañía, de manera constante y transparente para todos.

Objetivos Específicos

- Limitar el tiempo de parseo a 5 segundos por archivo XML como valor máximo.
- Customizar todas las etapas del procesamiento.
- Que el método de procesamiento sea el mismo, sin importar la estructura o el formato del archivo XML.
- Paralelizar el procesamiento sin afectar el resultado de las demás instancias.
- Mantener un formato de salida estándar por tipo de medición, sin importar la versión de la medición.
- Diseñar un método para procesar nuevos archivos XML que contengan nuevas estructuras nunca antes procesadas.
- Almacenar las estructuras de las mediciones, en un formato estándar y único sin importar que forma esta tenga.

Idea a defender | Propuesta a Justificar | Soluciones a comprobar

Un aspecto de las POC es que ninguna de las que se probaron dio los resultados esperados y que ni siquiera en la conjunción de las mismas se logra lo planeado. Las soluciones que más se acercan presentan impedimentos a mediano plazo como imposibilidad para futuras actualizaciones o para la definición de nuevos requerimientos.
Todas las pruebas de concepto están condicionadas por un proveedor, directa o indirectamente, es decir que todas las soluciones llevan a la contratación de un servicio y por lo tanto, los tiempos se extienden cosa que no es posible.
Una solución propia nos permite no depender de terceros, considerar de manera más puntual aspectos propios del modelo del negocio y solucionar aspectos como la escalabilidad, la implementación de futuras estructuras y de toda futura mejora.
Al momento de tomar la decisión, un componente importante que resulta relevante, es el conocimiento y uso de sentencias con AWK, que en situaciones muy similares ha funcionado muy bien, con buenos tiempos de respuesta y sin hardware de mas, con el que se tenía es suficiente, solo se necesita ingenio.
Otro factor importante es el tiempo que se tiene para la construcción de la solución, el cual es poco si se quiere capacitar en el negocio a quienes ofrecen los servicios de consultoría o a proveedores. El conocimiento y manejo del modelo del negocio, cualquier proveedor tiene primero que aprenderlo para luego proponer una solución.
Todas las soluciones demandan altos costos de hardware que luego se traducen en tiempos de puesta a punto de hardware comprado y también, la similitud en las soluciones propuestas, todos los “parser” son similares y dejan en evidencia los mismos problemas de escalabilidad de siempre.

Delimitación de proyecto

No se incluye ninguna de los siguientes aspectos:

- Uso de estructuras de control (if, for, case, etc.).
- Aspectos y usos de Sql Loader.
- Aspectos y consideraciones de base de datos.

Aporte Práctico

Las fuentes de información contienen datos sensibles de las redes de telecomunicaciones, qué proporcionados en tiempo y forma, sirven para la correcta toma de decisiones. Los principales consumidores de esta información son los especialistas de red, ellos construyen informes de manera semanal y mensual, indicando desde aspectos básicos hasta aspectos más detallados, incluso notificar desvíos propios del mal desempeño de la red.
Cualquier faltante de datos sea pequeño o grande tiene su impacto en los informes, el no contar con la información a tiempo o el tenerla incompleta generan un mismo resultado, incertidumbre en el comportamiento de la red y por lo tanto desconcierto en los especialistas.
Toda la compañía se beneficia de la implementación de una solución que presente los datos de la red en tiempo y forma, pero principalmente las gerencias de "Ingeniería" y "Calidad de Red" que son los responsables de la eficiencia de la red.
En cualquier tarea de optimización, construcción de reporte de desempeño o de resolución de un reclamo corporativo, todas tienen como punto de partida la fuente de información, es decir, los indicadores de desempeño de la red.
Por lo tanto, el soporte a la solución de un problema, cualquiera sea su índole es total y necesaria.

Aporte Teórico

La novedad del presente proyecto es la implementación de AWK en un entorno Unix, para el parseo de archivos XML de forma masiva.
Lo más novedoso es qué se trata de una solución a implementar en un equipo productivo que tiene una alta tasa de disponibilidad, es decir, con poco margen para el error.
Las pruebas conseguidas en la implementación de sentencias AWK sobre archivos XML, ha sido en definitiva, la mejor solución al problema.
Otro dato relevante es el gran volumen de archivos a procesar en un pequeño periodo de tiempo con una tasa de error mínima, sin picos de procesamiento en el servidor productivo. Por ejemplo, 5000 archivos XML en menos de una hora.
Todo el proceso es escalable y parametrizable en su totalidad, los aspectos que se consideran escalables tienen que ver con nuevas estructuras, nuevos formatos, nuevas presentaciones de archivos XML.
El procesamiento se adapta a cualquier estructura de archivo XML, siempre y cuando esta última este almacenada para su consideración dentro del proceso.

La elección e implementación de AWK como lenguaje central no solamente es producto de la experiencia adquirida en proyectos anteriores similares sino que también es por el potencial en si del lenguaje y de la posibilidad de indagar sobre nuevos métodos, nuevas técnicas sobre manejo de datos y en esto mejorar.
El descubrimiento de nuevos aspectos en AWK que podrían facilitar a otros proyectos existentes, como así también, un sinnúmero de escenarios que antes no era posible abordarlos y que hoy dejaron de ser un problema, es lo favorece la elección de dicho lenguaje.

Por ejemplo, la implementación de una máscara de fondo que permita a nivel de Unix, implementar un control de faltante de datos, es posible gracias a features descubiertos en AWK.

Método de investigación

Existen un par de elementos o de momentos dentro del proceso en el que es necesaria una investigación.
La estructura de la entrada del proceso Parser, en otras palabras, la estructura del archivo XML y sus partes. Se trata de una investigación *descriptiva* en donde se quiere conocer los elementos, ubicación y orden dentro del archivo, como así también las dependencias si existiesen.
Y el otro punto que es necesario investigar es la homologación de la salida de los archivos XML en archivos CSV.
AWK tiene la funcionalidad de poder entregar sus valores en arreglos (arrays) y este es el método más óptimo que se consideró para devolver valores. Para ello es necesaria una investigación *explorativa* del alcance de la funcionalidad y si se ajusta a los tiempos que se pretenden obtener.

Primera Parte. Marco Contextual.

Entorno del objeto de estudio

El objeto de estudio son los XML de Performance generados por el proveedor Nokia, los cuales se encuentran almacenados en tres servidores clusterizados, llamados "Regional Cluster".
Los Regional Cluster se encuentran clasificados por zona, cada uno tiene la información de la performance de la red de una zona geográfica del país, esto quiere decir que los XML en un Regional Cluster no se repiten en otro Regional Cluster y si se pierde, esa zona para ese periodo de tiempo, queda sin estadística de performance.
Para acceder a los distintos Regional Cluster, es preciso una conexión segura, una conexión de tipo SSH.

El ciclo de vida de un XML comienza al culminar la hora de la medición, por ejemplo, el XML con las mediciones de servicio de la hora 3:00 se crea a la hora 4:00 y queda disponible para su procesamiento durante 2 días y medio dentro de su Regional Cluster, pasado ese tiempo el XML se elimina liberando espacio, dependiendo del Regional Cluster será el tamaño del XML, pero en promedio cada XML tiene un tamaño de 2,5 Mbytes.

Frecuencia de generación y volumen de datos

Comencemos por la frecuencia, es donde hacemos referencia al periodo de disponibilidad de los datos, es decir, cada cuanto tiempo tengo disponible la información antes de que se genere una más nueva que la anterior. La frecuencia que manejan los datos de desempeño es de un XML por hora, por lo tanto recibiré nuevos XML con nuevos datos de desempeño, todas las horas.
Hablemos del volumen de datos y para esto tenemos que manejar dos nuevos conceptos, mediciones y elementos controladores de la red.

Mediciones

Las mediciones hacen referencia a la clasificación de lo que antes veníamos haciendo mención como "datos de desempeño de la red".
Los datos de desempeño de la red se clasifican en mediciones.
Veamos algunos ejemplos: Una medición puede hacer referencia a datos de las llamadas (cantidad de llamadas, cantidad de llamadas exitosas, total de minutos conmutados, etc…), otra medición puede hacer referencia al tráfico de datos (Mbytes que se descargan de la red, Mbytes que se subieron a la red, Ancho de banda disponible, tasa de transferencia, etc...) y otras que hacen referencia a la disponibilidad de la señal, intensidad y alcance.

NSN posee una gran variedad de mediciones y se pueden elegir, dependiendo de lo que se necesite medir o monitorear.
La jefatura tiene definidas algunas mediciones como básicas y que considera necesarias, para responder a todas las preguntas que se reciben sobre el desempeño de la red y que se utilizarían para la toma de decisiones son:

14 mediciones para medir el desempeño de la red 2G (GSM).
13 mediciones para el desempeño de la red 3G (UMTS).

8 mediciones para medir el desempeño de la red 4G (LTE).

Entre todas, hacen un total de 35 mediciones. Tanto UMTS como LTE son tecnologías que aún están creciendo por tanto el número de mediciones en el futuro puede ser mayor.

Elemento controlador de la red

Comencemos a explicar partiendo de una situación, una sencilla que podría ser la de efectuar una llamada desde un teléfono móvil, lo primero que tenemos que tener es señal, estar dentro del área de cobertura. El concepto de alcance de una señal de antena aplica perfectamente aquí, sino estamos dentro de la zona de cobertura de la antena que da la señal, no podremos establecer la comunicación. Esa antena se llama "celda" y se encuentran ubicadas casi siempre, en una torre o en alguna estructura que de soporte para ubicarlas en algún lugar alto y que permita tener mayor cobertura, como una antena de televisión que mientras más alto este, mejor señal recibe.
Las torres o las estructuras en donde se montan las celdas, las llamaremos "sitio".

Entonces, hasta el momento tenemos que las celdas se ubican en lo alto de un sitio y que puede haber más de una celda en un sitio. Las celdas generan hora por hora indicadores de desempeño, que son almacenados en un "elemento controlador", para su posterior análisis.
Todas las celdas, reportan hora por hora al elemento controlador datos sobre su funcionamiento y este es responsable a su vez de verificar la calidad de la señal de cada una de las celdas y de decidir cuál de todos los pedidos para establecer una llamada, es la que dará paso. El elemento controlador de la red, es el responsable del funcionamiento de todas las celdas que estén a su cargo.
El elemento controlador es el responsable también de la generación de las mediciones, es decir, al final de cada hora envía un XML midiendo el nivel de desempeño de sus celdas.

Si decimos que para la tecnología UMTS, existen alrededor de 125 elementos controladores generando 13 mediciones por hora cada uno, eso nos da un total de 1625 XML con mediciones por hora.
Cada XML tiene tantos nodos como celdas tenga el elemento controlador, porque allí se alojan, las mediciones de desempeño de cada celda. Un elemento controlador tiene en promedio alrededor de 400 celdas, por lo tanto, eso suma 50000 celdas por hora con datos de desempeño por hora por cada medición, para procesar.

Tabla de relaciones	
1 Tecnologia	esta compuesta por N Mediciones
1 Elemento Controlador	contiene a N Sitios
1 Sitio	contiene a N Celdas
1 Regional	almacena N archivos XML
1 archivo XML	contiene 1 o N Mediciones, 1 Elemento Controlador, 1 Hora
1 archivo XML	se traduce luego en 1 o N archivos CSV
1 archivo CSV	contiene 1 Medicion, 1 Elemento Controlador, 1 Hora

Tabla 1.1. Relaciones entre las distintas entidades objeto

Relación tesista y objeto de estudio

El área responsable de la extracción de los XML de Performance es la jefatura de Calidad de Red.
Mi puesto dentro del área es de "Especialista de Performance" y soy responsable de la extracción, procesamiento y posterior almacenamiento en base de datos de todos los datos existentes dentro de cada XML identificado para procesar en cada Regional Cluster.

Como mencionamos al comienzo la jefatura está en proceso de migración, es decir, está migrando de un modelo donde los datos son extraídos desde una BD a otro modelo donde la única fuente de información será un repositorio externo con muchos XML.

Análisis de los problemas observados

Presentación de los datos

La jefatura no cuenta con herramientas de gestión para el procesamiento de este tipo de archivos, es un escenario nunca antes visto.

Diversidad

Lo archivos XML se encuentran almacenados en los distintos Regional Cluster, a su vez, se clasifican por elemento de red y por tipo de medición.
Cada medición presenta una lista propia de contadores que puede ser de hasta 500 elementos.
La diversidad de valores y de posibles situaciones, no se había presentado nunca antes.

Frecuencia

Los archivos XML se generan por hora y contienen información de una medición y de un elemento controlador. Existen alrededor de 50 mediciones para un número de 100 elementos controladores aproximadamente, lo que hace un total aproximado de 5000 XML para procesar por hora.

Escalabilidad

Es un hecho innegable que los XML no solamente crecen en cantidad sino que también crecen en tamaño debido a tres factores:

Cantidad de contadores por medición

Las mediciones nunca conservan el mismo conjunto de contadores sino que tienden a enriquecerse de nuevos contadores, a medida que sufren actualizaciones o aparecen nuevos elementos de red con nuevas versiones de mediciones ya existentes.

Cantidad de elementos de red por elemento controlador

La cantidad de elementos que un elemento controlador concentra tiende a aumentar a medida que la red crece.
Muchas veces debido a la gran cantidad de elementos de red dentro de un controlador se toma la decisión de separar la cantidad de elementos en un nuevo elemento controlador.

Frecuencia

Este tercer factor y quizás es el más crítico, porque existe la posibilidad de que el nivel de granularidad deje de ser horario y pase a estar fraccionada la medición a 15 minutos.
Esto implicaría cuadruplicar el volumen de XML por hora y por el ende el volumen de XMLs a parsear.

Antecedentes de proyectos similares

El parseo de XML usando AWK no es nuevo y hay mucha información al respecto en internet.
La diferencia radica en el escenario actual, en los valores expresados en el punto anterior, que hacen al contexto único.
Todas las soluciones que se encuentren en Internet son a modo de ejemplo y no he encontrado ningún caso de éxito aplicando esta tecnología.
A nivel organizacional cualquier solución similar al proyecto en cuestión, todas vienen de la mano de Big Data por el volumen, granularidad y frecuencia de los datos.

Segunda Parte. Marco Teórico.

Marco Teórico del Objeto de Estudio

El objeto de estudio, lo hemos mencionado en anteriores puntos, es el XML. En este punto abordaremos los aspectos estructurales o de diseño que los XML presentan.

Open Measurement Standard (OMeS)

"OMeS" es el estándar usado por Nokia para el formato de los archivos XML.
Básicamente, es el formato estándar que se utiliza en los XML que salen de la base de datos de Nokia para alimentar sistemas externos.

Ahora explicaremos un poco más el modelo de datos de OMeS.

Elemento	Descripción	Ocurrencia	Contenido
OMeS	Es el elemento root de cada OMeS file	1	
PMSetup	Este elemento indica el comienzo de un nuevo periodo de medición. Define también el intervalo de tiempo y la hora de inicio de la medición mediante sus correspondientes atributos	1	
PMMOResult	Este elemento indica el inicio de un nuevo set de información.	1..*	
MO	Este elemento indica el comienzo de la definición topológica de un objeto de red. Es el primer elemento dentro del PMMOResult.	1..*	
DN	Este elemento se utiliza para especificar la información del objeto de red.	1	Puede contener el estándar de Nokia o el de 3GPP
PMTarget	El elemento indica el inicio de la información de la medición contenida en un set de contadores	1	
Counter	Este elemento define el valor individual de un contador. Puede presentarse varias veces dentro del PMTarget pero con ocurrencias siempre distintas. El nombre del elemento no puede contener carácter especial, solo están permitidos: underscore (_), hyphen (-) y dot (.)	1	El valor de cada contador es representado en formato decimal.

Tabla 2.1.1. Componentes del estándar OMeS

Como lo menciamos en varias oportunidades, el OMeS XML proviene desde los elementos de red hacia NetAct que es la solución en BD de Nokia para contener la información de los XML. Cada archivo contiene información del objeto medido, inicio e intervalo de la medición y lista de contadores.

Convención del nombre del archivo OMeS

El nombre de los archivos XML respetan por convención, el siguiente patrón:

```
etlexpmx_<firstMOClass>_<timeStamp>_<uniqueID>.xml.
```

Sección	Descripción
firstMOClass	Es la clase del objeto medido que se encuentra dentro del archivo con el listado de contadores de una medición.
timeStamp	Es el día y hora del momento en que se escribió el archivo en disco. No representa el inicio de la medición que está dentro.
uniqueID	Es un numero identificador, utilizado para referenciar un archivo de varios archivos que hayan sido creados en el mismo momento. Es un valor que se incrementa cada vez que se escribe un archivo en disco.

Tabla 2.1.2. Convención del esquema de nombres para los OMeS files.

Y para concluir con este punto, un ejemplo de un encabezado de un XML para representar gráficamente el modelo anteriormente mencionado, tanto del contenido del XML como de la convención del nombre.

```
/calidad/data/nsn/storage/xml/rc2/pm/2016110316># more etlexpmx_WCEL_20161103161257_3003864.xml
<?xml version="1.0"?>
<OMeS  xmlns="pm/cnf_rnc_nsn.7.0.xsd">
  <PMSetup  startTime="2016-11-03T15:00:00.000-03:00:00" interval="60">
    <PMMOResult>
      <MO>
        <DN><![CDATA[PLMN-PLMN/RNC-360/WBTS-35233/WCEL-35233]]></DN>
      </MO>
      <MO>
        <DN><![CDATA[PLMN-PLMN/MCC-722/MNC-310]]></DN>
      </MO>
      <PMTarget  measurementType="Cell_Throughput_WBTS">
        <M5002C0>4824</M5002C0>
        <M5002C1>5937</M5002C1>
        <M5002C10>0</M5002C10>
        <M5002C108>17525088</M5002C108>
        <M5002C11>0</M5002C11>
        <M5002C12>0</M5002C12>
        <M5002C125>1800055</M5002C125>
        <M5002C126>0</M5002C126>
```

Grafico 2.1.1. Contenido de un XML según convención OMeS.

Marco teórico del campo de acción

El campo de acción es el lenguaje de programación AWK. Para explicar esta tecnología comenzaremos contando brevemente su historia.

AWK, La historia.

AWK	
Información general	
Paradigma	Programación dirigida por eventos, Programación imperativa
Apareció en	1977, última revisión 1985, la versión POSIX actual es IEEE Std 1003.1-2004
Diseñado por	Alfred Aho, Peter Weinberger, and Brian Kernighan
Tipo de dato	débil, dinámico
Implementaciones	awk, GNU Awk, mawk, nawk, MKS AWK, Thompson AWK (compilador), Awka (compilador)
Dialectos	*old awk* oawk 1977, *new awk* nawk 1985, GNU Awk
Influido por	C, Bourne shell, SNOBOL
Ha influido a	Perl, Korn shell(*ksh93*, *dtksh*, *tksh*), JavaScript
Sistema operativo	Multiplataforma
[editar datos en Wikidata]	

AWK fue escrito originalmente en 1977 con UNIX.
En 1985 los autores comienzan la expansión del lenguaje, añadiendo funciones definidas por el usuario. El lenguaje es descrito en el libro *The AWK Programming Language*, publicado en 1988. Para evitar confusiones con la versión anterior, a la que era incompatible, esta versión se conoce, a veces, como "nuevo AWK" o "nawk". Esta implementación fue publicada bajo una licencia de software libre en 1996, continúa siendo mantenida por Brian Wilson Kernighan.

AWK es un lenguaje de programación diseñado para procesar datos basados en texto, ya sean ficheros o flujos de datos. El nombre AWK deriva de las iniciales de los apellidos de sus autores: Alfred **A**ho, Peter **W**einberger, y Brian **K**ernighan. awk, cuando está escrito todo en minúsculas, hace referencia al programa de Unix o Plan 9 que interpreta programas escritos en el lenguaje de programación AWK.

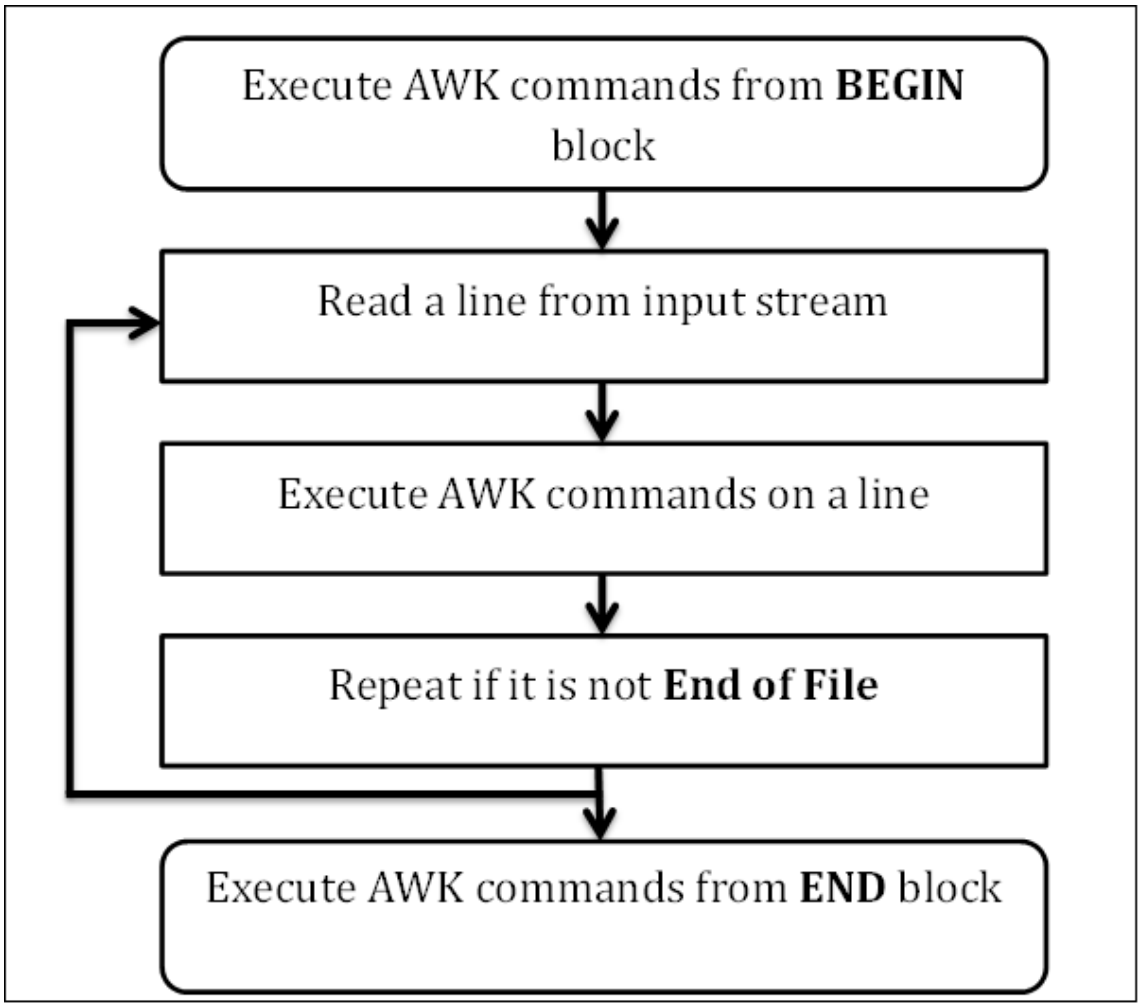
AWK es ejemplo de un lenguaje de programación que usa ampliamente el tipo de datos de listas asociativas (es decir, listas indexadas por cadenas clave), y expresiones regulares. El poder, brevedad y limitaciones de los programas de AWK y los guiones de sed inspiraron a Larry Wall a escribir Perl. Debido a su densa notación, todos estos lenguajes son frecuentemente usados para escribir programas de una línea.

AWK fue una de las primeras herramientas en aparecer en Unix (en la versión 3) y ganó popularidad como una manera de añadir funcionalidad a las tuberías de Unix. La implementación de alguna versión del lenguaje AWK es estándar en casi todo sistema operativo tipo unix moderno. AWK es mencionado en las Single UNIX Specification (especificaciones básicas de unix) como una de las utilidades necesarias de todo sistema operativo Unix. Se pueden instalar implementaciones de AWK en casi todos los demás sistemas operativos.

AWK es una herramienta muy útil para modificar archivos, buscar y transformar datos y, en general, realizar cualquier tipo de tratamiento masivo de ficheros.
Con un programa AWK es posible contar el número de líneas de un archivo, seleccionar columnas, aplicar filtros, realizar cruces, borrar el último campo de cada línea, hacer sumarizaciones, comprobar duplicados, muestreos, etc.

Estructura y ejecución

Un diagrama de flujo hace mucho más fácil la interpretación de este lenguaje de programación:



Grafica 2.2.1. Diagrama de Flujo de como procesa las sentencias AWK.

Los programas AWK poseen tres bloques:
Uno que se ejecuta una sola vez al comienzo del programa (bloque BEGIN).
El segundo que es el procesamiento en sí, donde se ejecutan todas las instrucciones escritas, una vez por cada línea del archivo a procesar.
Tercero y último, un bloque que solo se ejecuta una vez al final del programa (bloque END).

Un programa AWK presenta la siguiente estructura de reglas:

```
patrón { acción }
patrón { acción }
...
```

Los patrones corresponden a expresiones regulares (por ejemplo /foo/) o condiciones (por ejemplo $1 = “Jan”) y las acciones son funciones del lenguaje.
En una regla puede omitirse el patrón o la acción, pero no ambos. Si el patrón se omite, entonces la acción se realiza para todas las líneas. Si se omite la acción, la acción por defecto es imprimir las líneas que cumplan el patrón.

fichero_A.txt

```
        Jan 13 25 15 115
        Feb 15 32 24 226
        Mar 15 24 34 228
        Apr 31 52 63 420
        May 16 34 29 208
```

```
Jun 31 42 75 492
Jul 24 34 67 436
Aug 15 34 47 316
Sep 13 55 37 277
Oct 29 54 68 525
Nov 20 87 82 577
Dec 17 35 61 401
Jan 21 36 64 620
Feb 26 58 80 652
Mar 24 75 70 495
Apr 21 70 74 514
```

Por ejemplo, para las lineas que contienen “Jan” las imprimimos por pantalla (simulamos comando ‘grep’):

```
awk '/Jan/ { print $0 }' fichero_A.txt
```

Si quisiéramos únicamente imprimir la segunda columna utilizamos $1 (el resto las podemos encontrar en $2, $3, etc.):

```
awk '/Jan/ { print $1 }' fichero_A.txt
```

También podemos realizar sumarizaciones, por ejemplo, sumamos los valores de la quinta columna de las lineas que tengan en la primera columna “Jan”:

```
awk '$1 == "Jan" { sum += $5 } END { print sum }' fichero_A.txt
```

Como se puede observar, se han definido 2 reglas:

- Si la primera columna es “Jan” entonces acumulamos el quinto valor en una variable.
- Y al final del fichero (END {…}), imprimimos el valor de la variable.

Es muy habitual ver ejemplos de AWK en una única línea, y justamente por ese motivo se ha ganado la fama de complejo. Pero en realidad podemos crear ficheros de texto con todas las instrucciones bien tabuladas y que permiten una mejor comprensión del objetivo del programa. Por ejemplo, podemos crear el fichero ‘test.awk’:

```
#!/usr/bin/awk -f
$1 == "Jan" { sum += $5 }
END { print sum }
```

Para ejecutarlo, utilizaremos la siguiente sentencia:

```
awk -f test.awk fichero_A.txt
```

El primer argumento siempre corresponde al fichero sobre el cual vamos a ejecutar el programa. De hecho, desde el propio programa podemos acceder al listado de argumentos con el que se ha llamado:

```
#!/usr/bin/awk -f
BEGIN {
        print ARGC      # Número de argumentos
        print ARGV[0]   # Siempre "awk"
        print ARGV[1]   # Primer argumento
        print ARGV[2]   # Segundo argumento
}
```

Incluso, podríamos modificar los argumentos desde el propio programa para que no sea necesario especificar los ficheros por linea de comandos:

```
#!/usr/bin/awk -f
BEGIN {
  ARGC = 2 # Uno más que el número de ficheros de entrada que indiquemos.
  ARGV[1] = "file1"
}
{
  print $0
}
```

Separadores de registros

AWK realiza un tratamiento registro a registro. Por defecto los registros se encuentran delimitados por el final de línea \n y por tanto, un registro = una línea. No obstante, podemos cambiar el símbolo que denota el final de un registro a otro carácter:

```
    #!/usr/bin/awk -f
        BEGIN {
                ARGC = 2 ; ARGV[1] = "fichero_A.txt"
                RS = "1"
        }
        { print $0 }
```

En el anterior programa, hemos cambiado la variable RS de forma que AWK compondrá el primer registro leyendo todos los caracteres hasta encontrar un “1” y así sucesivamente.
¿Qué utilidad tiene este mecanismo? Por ejemplo, en caso de que tengamos un fichero donde los registros se encuentren separados por líneas en blanco como el siguiente:

fichero_cortado.txt

```
        Primer_registro dato1 dato2
        dato3 dato4

        Segundo_registro dato1 dato2

        Tercer_registro dato1 dato2 dato3
```

Podemos establecer RS = “”, entonces awk compondrá los registros desde el primer carácter hasta que encuentra una línea en blanco. Veámoslo con un ejemplo:

```
        #!/usr/bin/awk -f
        BEGIN {
                ARGC = 2 ; ARGV[1] = "Fichero_Cortado.txt"
                RS = ""
        }
        { print FNR " " NR " " $1 " " $2 " " $3 " " $4 " " $5 }
```

El anterior programa nos reformatearía el archivo a:

```
        1 1 Primer_registro dato1 dato2 dato3 dato4
        2 2 Segundo_registro dato1 dato2
        3 3 Tercer_registro dato1 dato2 dato3
```

Hemos utilizado la variable FNR que indica el número de registro del fichero actual y NR que hace referencia al número de registro total (solo tiene sentido en caso de que hayamos utilizado varios ficheros de entrada).

Separadores de campos y columnas de tamaño fijo

Por defecto AWK identifica las columnas utilizando como separador el espacio o el tabulador (si hay varios consecutivos son ignorados). Es posible modificar este separador mediante la variable FS:

```
        #!/usr/bin/awk -f
        BEGIN { FS = ";" }
```

Incluso podríamos utilizar expresiones regulares para indicar delimitaciones:

```
        FS = ", \t" # Coma y tabulador
        FS = " " # Por defecto
        FS = "[ ]" # Un único espacio, si hay varios se consideraran campos
vacíos
```

En el caso de que queramos modificar el delimitador de registros y columnas para la salida utilizaremos las variables ORS y OFS respectivamente:

```
        #!/usr/bin/awk -f
        BEGIN {
           ARGC = 2 ; ARGV[1] = "fichero_A.txt"
           FS = " " # Delimitador columnas
           OFS = ";"
           RS = "\n" # Delimitador registros
           ORS = "[FIN]\n"
        }
        { print $1, $2, $3 }
```

Para la interpretación de ficheros que no disponen de delimitadores, sino que tienen columnas de tamaño fijo como por ejemplo:

Fichero_ColumnasFijas.txt

```
        937563.51176.9000.0043.9600.0801
        937663.52276.9060.0043.9620.0801
        937763.51476.9360.0043.9660.0801
        937863.53776.9240.0043.9670.0801
        937961.96778.5430.0044.1090.0801
        938061.95178.5420.0044.1080.0801
```

Podemos utilizar la variable FIELDWIDTHS, donde especificaremos en orden el tamaño de cada uno de los campos:

```
        #!/usr/bin/awk -f
        BEGIN {
            ARGC = 2 ; ARGV[1] = "Fichero_ColumnasFijas.txt"
            FIELDWIDTHS = "4 6 6 5 5 5 1"   # Tamaño de cada campo en orden
        }
        { print $1, $2, $3, $4, $5, $6, $7 }
```

Para hacer referencia al registro completo hemos utilizado $0, pero si queremos acceder a columnas concretas se utiliza $1, $2,… $NF (variable que hace referencia al último). Cabe mencionar que si la columna no existe, no se produce ningún error… simplemente se muestra una cadena vacía.

Si bien $NF apunta a la última columna, NF nos indica el número de columnas que tiene el registro. Podemos utilizar ese valor para acceder a un campo anterior. El siguiente ejemplo nos mostraría el penúltimo campo de cada registro:

```
#!/usr/bin/awk -f
BEGIN { ARGC = 2 ; ARGV[1] = "fichero_A.txt" }
{ print $(NF-1) }
```

De hecho, podemos utilizar $() para indicar dentro de los paréntesis cualquier tipo de operación (por ejemplo $(2*2-1))

Campos computados

Los campos, además de ser mostrados, pueden ser modificados directamente o incluso creados (por ejemplo campos computados):

```
#!/usr/bin/awk -f
BEGIN { ARGC = 2 ; ARGV[1] = "fichero_A.txt" }
{ $2 = $2 - 10 }
{ $(NF+1) = "Nuevo campo" }
{ print $0 }
```

Agrupación de acciones

No es necesario tener una regla para cada acción ({acción} {acción} …), sino que es posible agrupar diversas acciones en una única regla:

```
#!/usr/bin/awk -f
BEGIN { ARGC = 2 ; ARGV[1] = "fichero_A.txt" }
{
        $2 = $2 - 10
        $(NF+1) = "Nuevo campo"
        print $0
}
```

O podemos poner todas las acciones en la misma línea pero separadas por “;”, aunque esta opción complica la lectura del código:

```
#!/usr/bin/awk -f
BEGIN { ARGC = 2 ; ARGV[1] = "fichero_A.txt" }
{ $2 = $2 - 10 ; $(NF+1) = "Nuevo campo" ;  print $0 }
```

Ficheros de salida

Hasta el momento hemos utilizado la acción “print” para mostrar registros por pantalla, si quisiésemos tener un mayor control del output podríamos utilizar “printf” de forma muy similar a como se realiza en el lenguaje C:

```
#!/usr/bin/awk -f
BEGIN { ARGC = 2 ; ARGV[1] = "fichero_A.txt" }
NR == 1 {
    printf "Entero %i\n", $1
    printf "Decimal %f\n", $1
    printf "Caracter ASCII %c\n", $1
    printf "Notación exponencial %e\n", $1
```

```
            printf "Cadena %s\n", $1
            printf "Octal %o\n", $1
            printf "Hexadecimal %X\n", $1
            # Modificadores
            printf "Cadena a la derecha en columna de 10: %10i\n", $1
            printf "Cadena a la izquierda en columna de 10: %-10i\n", $1
            printf "Número con 2 decimales %.2f\n", $1
        }
```

Como hemos visto, tanto print como printf escriben por defecto a la salida estándar. Quizás nos interese hacer que la salida se escriba en un fichero o se redirija a un comando mediante una pipe:

```
#!/usr/bin/awk -f
BEGIN { ARGC = 2 ; ARGV[1] = "fichero_A.txt" }
{
    print $0 > "001_fichero_salida.txt" # Fichero nuevo (borra si ya existia)
    print $0 >> "001_fichero_acumulativo.txt" # Append si el fichero existe
    print $0 | "/bin/cat" # Pipe a un comando
    print $0 | "sort -r > 001_fichero_salida.ordenado.txt"
}
END {
close("/bin/cat");
close("sort -r > 001_fichero_salida.ordenado.txt")
}
```

Como se observa en el ejemplo anterior, podemos aprovechar las ventajas de las pipes para combinar la potencia de AWK con otros comandos como “sort”. Hay que tener en cuenta que el comando no se ejecuta hasta que la pipe no es cerrada con close o por el fin del programa, esto implica que toda la salida se guarda en memoria mientras tanto.

Expresiones

Operadores aritméticos
Suma: x+y
Resta: x-y
Negación: -x
Multiplicación: x*y
División: x/y
Resto: x%y
Exponente: x**y
Comparaciones: x<y , x&#60=y, x>y, x>=y, x==y, x!=y, x~y (‘y’ debe ser una expresión regular, por ejemplo /^test/)
Expresiones booleanas: &&, ||, ! (negación)

Conversiones de formato

Las conversiones entre números y cadenas son automáticas (en caso de que una cadena no pueda ser convertida se traduce como 0):

```
        #!/usr/bin/awk -f
        BEGIN {
            edad = "27"
            edad = edad + 1.5
            print "Tengo " edad " años."
        }
```

En el ejemplo anterior edad es inicializada con una cadena, a continuación se utiliza en una suma y se convierte automáticamente a numérico.
Finalmente print convierte el número a cadena para permitir la concatenación.

Control del flujo

Las estructuras de control de awk son muy parecidas a las del lenguaje C:

```
#!/usr/bin/awk -f
BEGIN {
        # Condicional
        edad = "27"
        if (edad == 27) {
                print "Si"
        } else {
                print "No"
        }

        # Bucles
        i = 1
        print "While"
        while (i < = 3) {
                print i
                i++
        }

        print "Do while"
        do {
                i--
                print i
        } while (i > 1)

        print "Bucle for"
        for(i=1; i< = 3; i++) {
                print i
        }
}
```

En los bucles podemos utilizar "continue" para saltar a la siguiente iteración o "break" para finalizar. A nivel de programa, como AWK ejecuta todas las reglas para cada registro de los ficheros de entrada, en cierta forma se trata de un bucle que podemos hacer saltar al siguiente registro con "next" o parar con "exit".

Arrays asociativos

Como estructura de almacenamiento de información ya hemos trabajado con variables, pero AWK nos ofrece la posibilidad de usar arrays asociativos (diccionarios):

```
#!/usr/bin/awk -f
BEGIN {
        v[0] = "Elemento"
        v[1] = "Cadena"
        v["Clave"] = "Valor"
        v["key"] = 25

        # Comprobar la existencia de una clave
        if ("Clave" in v) {
                print "Clave encontrada!"
        }
```

```
                # Borrado de elementos
                delete v["Clave"]

                # Recorrer un array
                for(key in v) {
                        print key " -> " v[key]
                }

                # Arrays multidimensionales
                w[0, 1] = ";-)"
        }
```

Otras acciones/funciones

Veamos otras acciones útiles de awk:

```
#!/usr/bin/awk -f

BEGIN
{

## Números
srand() # Inicializa semilla con la fecha/hora actual
print rand() # Decimal aleatorio entre 0 y 1
print int(2.45) # Parte entera
num = sprintf("%.2f", 2.256821)   # Redondeo a 2 decimales
print num
print sqrt(4) # Raiz cuadrada
print "Exponencial " exp(1) " sinus " sin(2) " cosinus " cos(2) " logaritmo "
log(2)

## Strings

# Posición donde se encuentra "de" (0 si no lo encuentra)
print index("cadena", "de")
print length("cadena") # Tamaño
print match("cadena", /$.*/) # Compara con expresión regular
str = "1;2;3"
split(str, v, ";")  # Divide y guarda en un vector/array
print v[1] " " v[2] " " v[3]
sub(/;/, "-", str) # Substituye la primera aparición ";" por "-"
print str
gsub(/;/, "-", str) # Substituye todos los ";" por "-"
print str

# Desde la posición 1, devuelve los siguientes 3 caracteres
print substr(str, 1, 3)
print tolower("MAYUSCULAS")
print toupper("minusculas")

#Fechas y horas
t1 = systime()  # Fecha/hora actual
t2 = mktime("2009 01 01 00 00 00") # YYYY MM DD HH MM SS
print "Tiempo transcurrido desde 01-01-2009: "
print "- Segundos: " t1 - t2
print "- Minutos: " (t1 - t2)/60
print "- Horas: " (t1 - t2)/60/60
print "- Dias: " (t1 - t2)/60/60/24

# Formatear hora/fecha actual
print strftime("%Y-%m-%d %H:%M:%S", systime()) # 2009-02-14 17:46:28

}
```

Para el formateo de fechas se utiliza la siguiente nomenclatura:

```
        %a      %A      Nombre del dia (por ejemplo Lunes)
        %b      %B      Nombre del mes
        %d              Día del mes
        %H              Hora (24)
        %I              Hora (12)
        %j              Día del año (001-365)
        %m              Mes
        %M              Minuto
        %p              AM/PM
        %S              Segundo
        %u              Número del día de la semana (lunes = 1)
        %U              Número de la semana del año
        %Y              Año
```

Funciones

AWK nos permite definir nuestras propias funciones:

```
        #!/usr/bin/awk -f
        function p (str) {
                print str
                return 0    # Opcional
        }

        BEGIN {
                p("Hola mundo!")
        }
```

Esto nos permitiría hacer funciones genéricas que nos ahorren código repetitivo.

Expresiones regulares

```
^       Inicio de la cadena
$       Final de la cadena
.       Cualquier carácter único (excepto newline)
[]      Conjunto de caracteres
        [MVX]   Encaja con M, V o X
        [0-9]   Encaja con un dígito
[^]     Conjunto de carácter complementario
        [^0-9]  Encaja con cualquier carácter excepto un dígito
|       Alternativas
        A|[0-9] Encaja con "A" o un dígito
()      Se pueden utilizar paréntesis para agrupar expresiones
        (A|[0-9])Z      Encaja con "A" o dígito, que uno u otro vaya seguido de
Z
*       La expresión precedente se puede repetir 0 o más veces
        A*      Encaja con nada o un número indefinido de A
+       La expresión precedente tendrá lugar 1 o más veces
        A+      Encaja con una o más A
?       La expresión precedente no aparecerá o aparecerá 1 única vez
```

Para que AWK no sea sensible a las mayúsculas/minúsculas podemos jugar con las acciones tolower/toupper o bien establecer la variable IGNORECASE a 1.

Nombres de columnas

Por defecto, AWK nos permite acceder a las columnas mediante el uso de $1, $2, etc… pero esto puede ser un inconveniente si cambia el número u orden de las columnas de nuestros

ficheros, dado que implicará un cambio forzado en las referencias que usamos en nuestro programa. Para solucionar ese problema, si disponemos de ficheros donde la primera línea es la cabecera y se anotan los nombres de las columnas, podemos crear un array que después nos permita acceder como sigue:

```
        #!/usr/bin/awk -f
        BEGIN {
                ARGC = 1+1 # Uno más que el número de ficheros de entrada que
indiquemos.
                ARGV[1] = "Fichero_Column.txt"
        }
        # Cabecera
        NR==1 {
                        # Guardamos nombres de columnas
                        for(i = 1; i< = NF; i++) {
                                c[tolower($i)] = i
                        }
                 }

        # Podemos acceder a las columnas por $1, $2... o por
$c["nombre_columna"]
        NR > 1 { print $c["mes"] }
```

De esta forma podremos acceder a las columnas usando $c[“nombre_columna”], independientemente que en el futuro la cambiemos de lugar en el fichero de entrada.

Sumarizaciones

Para realizar una sumarización de un fichero por el campo $1 y sumando los valores del campo $3, podemos utilizar el siguiente código (no requiere que el fichero este ordenado):

```
        #!/usr/bin/awk -f
        {
                # Si no es la primera vez que tratamos esta clave...
                if (count[$1] != "") {
                        count[$1]++
                        col_sum[$1] += $3
                } else {
                        count[$1] = 1
                        col_sum[$1] = $3
                }
         }

        END {
                for (k in count) {
                        print k " " count[k] " " col_sum[k]
                }
        }
```

Podríamos añadir más arrays para hacer más sumarizaciones de columnas, o utilizar como índice más campos (por ejemplo count[$1, $2]) si queremos utilizar varias columnas como campos clave.
Por supuesto, con este mismo ejemplo podríamos validar si existen duplicados dado que en “count” estamos guardando el número de apariciones.

Diagnóstico

Las implementaciones que se encontraron sobre el tema fueron sólo a nivel de ejemplo y nunca como parte de un proyecto productivo.
La implementación de AWK como herramienta de trabajo resultaba efectiva pero sólo en la teoría y una de las desventajas que posee AWK es ser considerado un programa "poco amigable", es decir, de muchas líneas para la concreción de un resultado.

Con esto en escena, podemos concluir que no existe evidencia suficiente para confirmar qué AWK es la mejor manera de procesar un OMeS file o que exista en la actualidad algún proceso que utilice dicha tecnología para el procesamiento de archivos OMeS.

La construcción de la solución se encaminó en base a la evidencia funcional de AWK y usando como fundamento las potencialidades descriptas en referencia al lenguaje de AWK y no así en soluciones productivas las cuales se cité como marco de estudio el lenguaje de AWK, tales soluciones no se encontraron en internet a la fecha.

Tercera Parte. Modelo Teórico.

Planificación

Para este punto se utilizó una herramienta de gestión gratuita y se tuvo en cuenta los plazos acordados y cada uno de los sprints que se definieron en el anteproyecto.
El objetivo al finalizar cada Sprint es entregar una parte del proceso global, una funcionalidad, gracias al acoplamiento existente entre los procesos es posible las entregas preliminares.

En cuanto a las actividades del proyecto, se intentó ser lo más específico y detallista posible. La idea principal de la definición de la actividad es poder trazar los tiempos y los esfuerzos que concentraron y requirieron cada ítem del proyecto su totalidad para evitar cuellos de botella o situaciones de conflicto en los próximos sprints.

Aquí un detalle a macro nivel del proyecto y sus tiempos:

Etapas	Fecha Desde	Fecha Hasta	Duracion en dias
1. Conexion SFTP	01/08/2016	12/08/2016	12
2. Construccion Lista	15/08/2016	26/08/2016	12
3. Primer Programa AWK	29/08/2016	09/09/2016	12
4. Segundo Programa AWK	12/09/2016	23/09/2016	12
5. Unificacion de datos	26/09/2016	30/09/2016	5

Tabla 3.1.1 Detalle Macro del proyecto, etapas y tiempos.

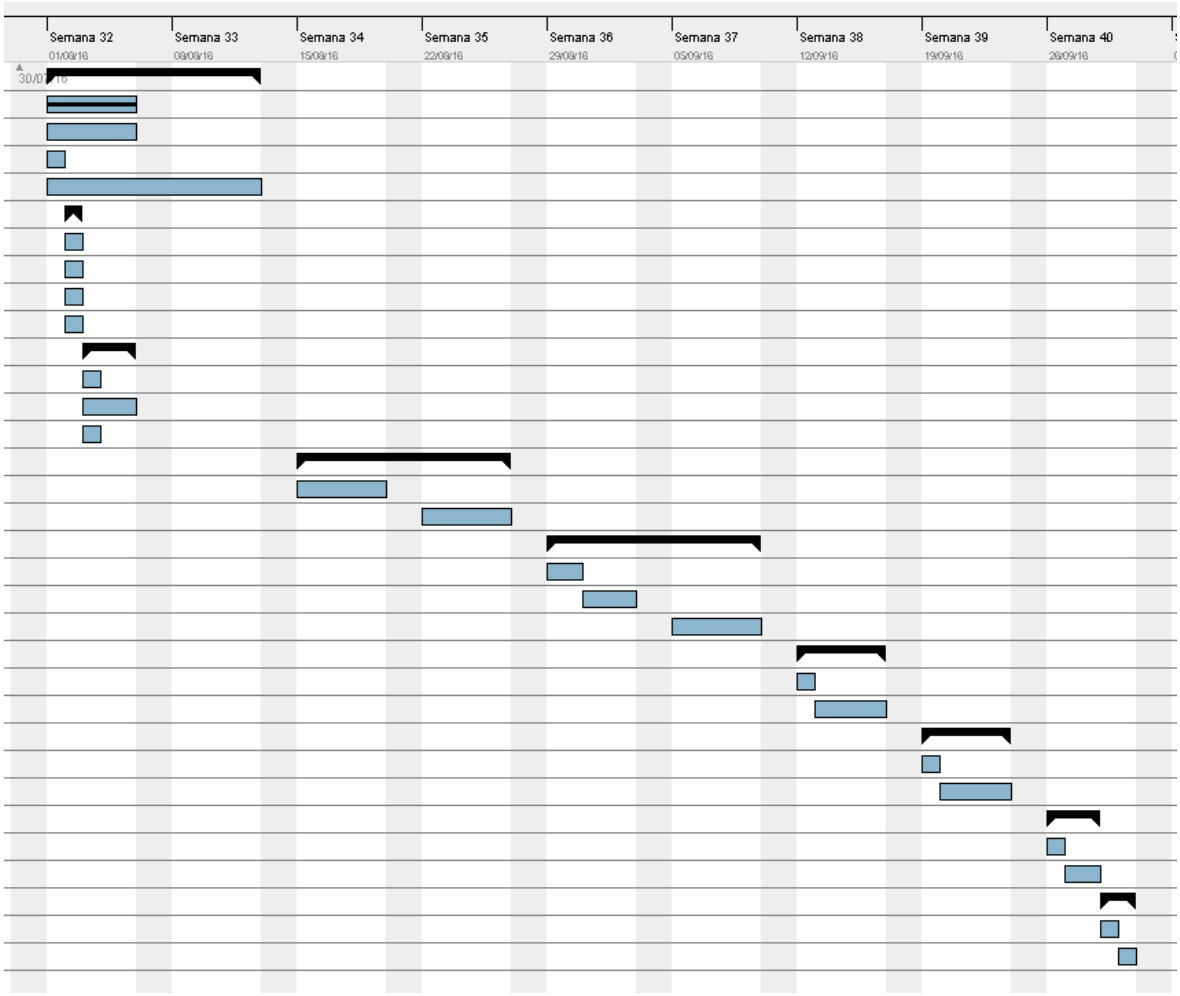
El proyecto consta de dos meses de trabajo repartidos en 5 etapas. Cada etapa representa un Sprint y termina con un entregable, una funcionalidad completa o estable y que es necesaria dentro del proceso parseador.

En un detalle a nivel de actividad se puede apreciar más la carga de trabajo por actividad:

Nombre	Fecha de inicio	Fecha de fin
Gestion de usuarios y contraseñas, ubicacion de archivos XML en servidores remotos	01/08/16	12/08/16
Solicitar IP o nombre de cada regional	01/08/16	05/08/16
Solicitar Usuario y Constraseña por Regional	01/08/16	05/08/16
Construccion de claves publicas y privadas	01/08/16	01/08/16
Proceso de Backup desde Storage remoto a Storage Interno	01/08/16	12/08/16
Implementacion de ambiente seguro para autenticacion sin contraseña	02/08/16	02/08/16
Construccion de carpeta .ssh por regional	02/08/16	02/08/16
Copiado de authentication file en cada .ssh	02/08/16	02/08/16
Asignacion de permisos 644 en cada .ssh	02/08/16	02/08/16
Identificacion de metodo de encriptacion y pass phase	02/08/16	02/08/16
Proceso de descompresion y otorgamiento de permisos sobre carpeta con XML en Storage Interno	03/08/16	05/08/16
Identificacion de carpeta en destino	03/08/16	03/08/16
Construccion de sentencia de descompresion por carpeta	03/08/16	05/08/16
Otorgamiento total de permisos para el procesamiento de los archivos	03/08/16	03/08/16
Proceso de Identificacion de XMLs a procesar	15/08/16	26/08/16
Construccion de lista de elementos de red	15/08/16	19/08/16
Construccion de lista de mediciones por elemento de red	22/08/16	26/08/16
Proceso Parser | Primera Parte	29/08/16	09/09/16
Identificacion del XML File	29/08/16	30/08/16
Construccion dinamica de la referencia a elemento a medir (Fecha, Hora, RC, NEC, Sitio, Celda)	31/08/16	02/09/16
Trasposicion de nodos con sus atributos en filas	05/09/16	09/09/16
Proceso Parser | Segunda Parte	12/09/16	16/09/16
Identificacion de lista de contadores segun medicion	12/09/16	12/09/16
Ordenamiento de contadores segun lista preestablecida	13/09/16	16/09/16
Proceso de Invocacion al proceso Parser	19/09/16	23/09/16
Identificacion de parametros de entrada	19/09/16	19/09/16
Encapsulamiento de procesos parser (primera y segunda parte)	20/09/16	23/09/16
Proceso de Unificacion de archivos de salida	26/09/16	28/09/16
Proceso de identificacion de archivos procesados	26/09/16	26/09/16
Unificación de archivos por elemento de red y por medicion	27/09/16	28/09/16
Proceso de persistencia en BD	29/09/16	30/09/16
Identificacion de archivos unificados (elemento de red, medicion, ruta de ubicacion)	29/09/16	29/09/16
Almacenamiento de los nombres de los archivos unificados en tabla de BD, para posterior tratamiento	30/09/16	30/09/16

Tabla 3.1.2 Detalle a nivel de actividad del proyecto, etapas y tiempos.

Diagrama de Gantt de las etapas y sus tiempos:

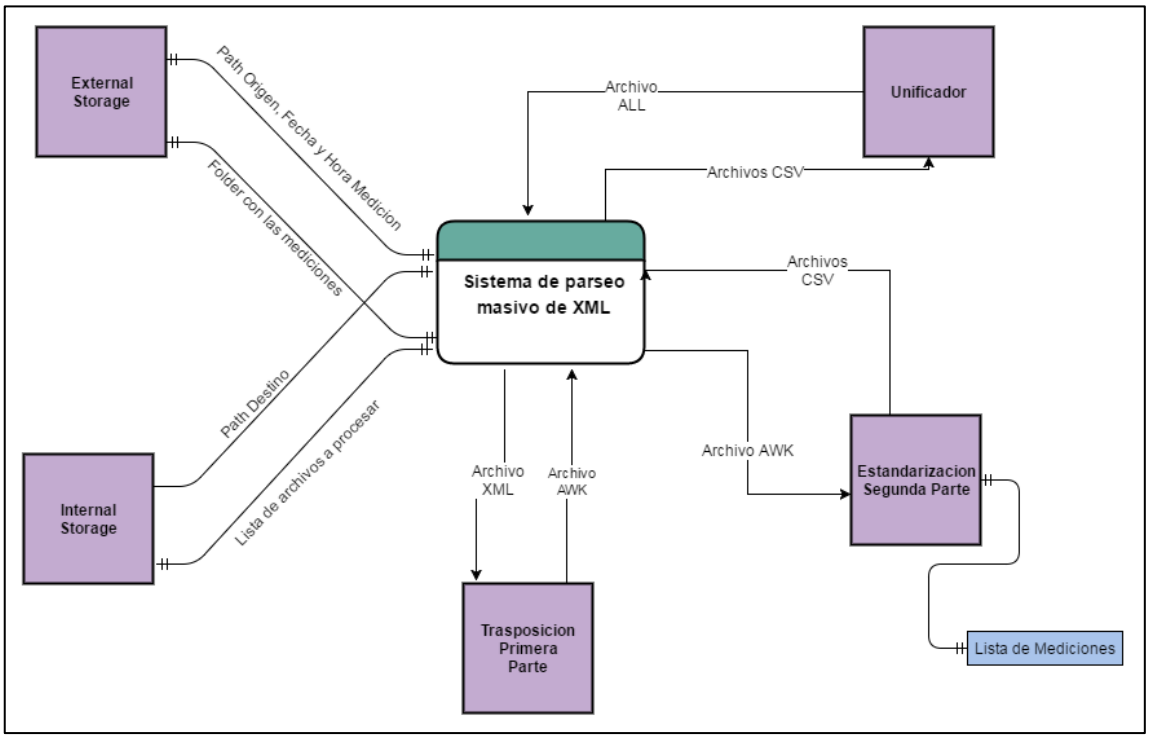


Grafica 3.1.1 Diagrama de Gantt de las actividades del proyecto.

Este es el plan ideal, el estimado. Esperamos no atrasar los tiempos y llegar a la solución dentro de los plazos acordados.

Requerimientos

Diagrama de contexto



Grafica 3.2.1. Diagrama de Contexto de la Solución Parser.

Este es el diagrama general del proyecto y también se podría decir que es clave para comprender y visualizar la dimensión del proyecto en sí.

El diagrama contiene 5 componentes claves:

- ✓ Elemento External Storage.
- ✓ Elemento Internal Storage.
- ✓ Componente de Transposición.
- ✓ Componente de Estandarización.
- ✓ Componente Unificador.

Cada uno tiene una función y un rol dentro del sistema.

Elemento External Storage

Hace referencia al proceso de conexión al Regional Cluster destino para extracción de datos. Utiliza dos parámetros: uno es el Regional Cluster y el otro es la hora que se desee recolectar y extraer para su posterior tratamiento.

Restricciones
Se puede mencionar como restricción que es necesaria la existencia de un certificado de confianza entre ambos servidores para que se establezca de manera automática y sin autenticación una conexión desde el proceso.

Limitaciones
Si por algún motivo el proceso de recolección de estadísticas se interrumpiera y fuese necesario tener que ejecutarlo nuevamente, no continuara desde donde termino sino que es preciso antes eliminar en el servidor destino la carpeta con todos los datos de la hora.
Es necesaria esta operación para identificar el faltante de datos, se realiza a nivel de hora y no por cantidad de XML dentro de la carpeta.

Elemento Internal Storage

Hace referencia al proceso de identificación de archivos XML a procesar. Recibe como parámetro de entrada la hora que se desea procesar y como resultado final, se obtiene un listado de archivos XML que tienen las mediciones que se necesitan procesar.

Restricciones
Como el resultado final es una sola lista y no una lista por elemento de red, es necesario esperar al que el proceso anterior termine en su totalidad para construir la lista.

Limitaciones
La lista es un factor condicionante en los próximos procesos, es decir, XML que no esté en la lista, XML que no se procesa.
No se puede tener una lista por elemento de red, por tanto, en caso de conocer que la lista está incompleta se debe borrar y comenzar a construirla nuevamente.

Componente de Transposición

En este primer componente del proceso de parseo y el objetivo es transponer los valores del XML que se encuentran encolumnados y detallados uno debajo de otro, a un formato en fila obteniendo algo más parecido a una tabla, un registro por fila y almacenarlos para su posterior tratamiento.
Es un componente que se basa en las reglas de la convención OMeS y es de iterativo e incremental. No está condicionado ni al largo de la lista de contadores, ni a la cantidad de nodos que posea el XML. El componente está apto para ejecuciones en paralelo.

Restricciones
No posee restricciones.

Limitaciones
No posee limitaciones.

Componente de Estandarización

Este el segundo y último componente que culmina con el proceso de parseo. Tiene como entrada la salida del componente anterior y una lista homologada de contadores por medición.
El objetivo es estandarizar una salida y construir un único formato estándar sin importar la medición, periodo de sumarización, elemento de red o Regional Cluster destino.

Restricciones
No posee restricciones.

Limitaciones
No posee limitaciones.

Componente Unificador.

La salida del componente anterior puede ser uno o varios archivos con los valores del XML pero separados por coma (archivos CSV) listos para ser insertados en la BD.
Pensando en el proceso de inserción en BD, se considera buena práctica el hacer una sola inserción masiva y no hacer muchas inserciones en BD.
Tomando como punto de partida que el formato del archivo CSV ya está estandarizado, es posible crear un solo archivo por medición. Este archivo tiene por extensión .all.

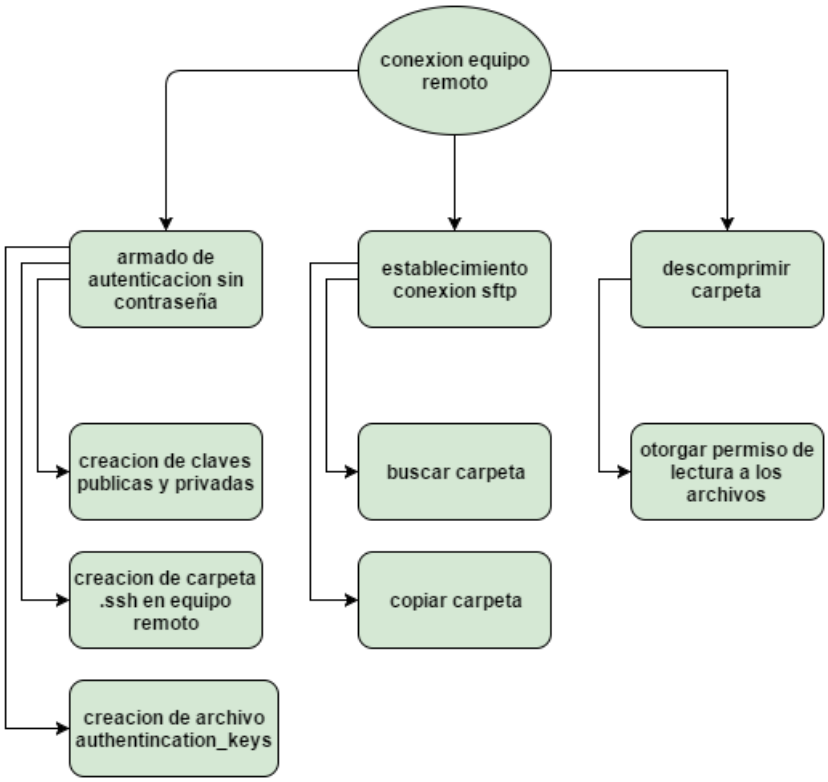
Restricciones
No posee restricciones.

Limitaciones
No posee limitaciones.

Análisis y Diseño

Diagramas de descomposición funcional de los subprocesos existentes

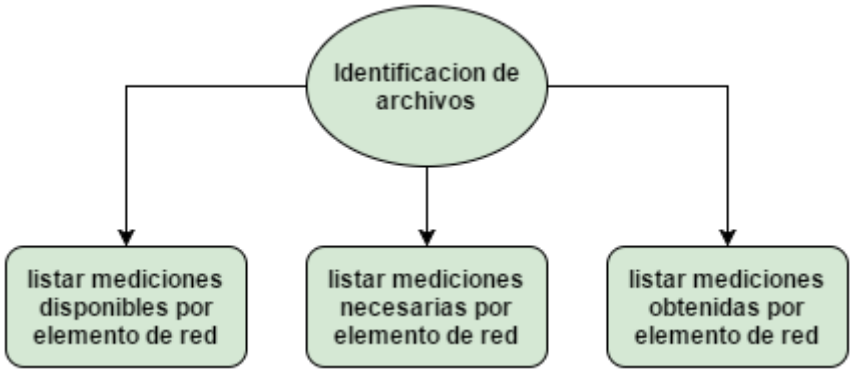
Diagrama del Proceso de Conexión a Servidor Remoto



Grafica 3.3.1. Diagrama de Descomposición Funcional. Proceso de Conexión y de Recolección de archivos.

Este el primer proceso que se ejecuta y es el responsable del copiado de los archivos XML desde el servidor origen (Regional Cluster) al servidor destino (Internal Storage).
Para ello, es necesario la construcción de un certificado de confianza entre el Regional Cluster y el Internal Storage.
Para el proceso de copiado de archivos es preciso indicar la fecha y hora que se quiere copiar para procesar.
Por último, se descomprimen los archivos y se otorgan permisos de escritura.
Los permisos de escritura no son para escribir dentro del XML sino que una vez procesado el archivo este se almacena por un periodo de tiempo y luego es eliminado. Para la eliminación, es necesario tener permiso de escritura sobre los archivos.

Diagrama de identificación de archivos

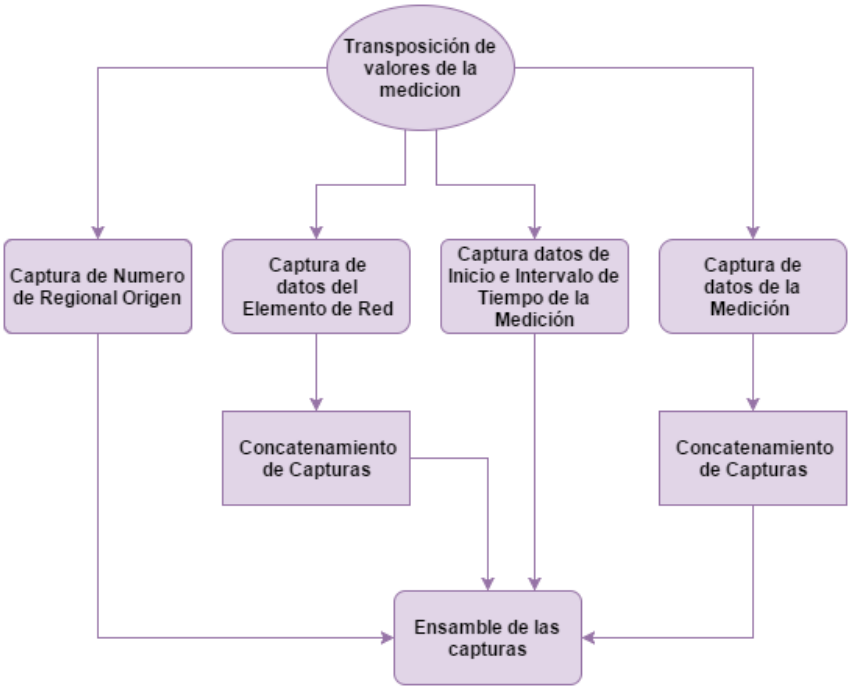


Grafica 3.3.2 Diagrama de descomposición funcional. Proceso de Identificación de archivos.

El proceso de identificación de archivos a procesar se realiza una vez descomprimidos los archivos XML en el Internal Storage.

El proceso consta de tres acciones bien claras:
Primero, armar un listado de todas las mediciones por archivo XML.
Segundo, cargar el listado de todas las mediciones necesarias.
Tercero y último, armar un listado de los archivos obtenidos que tienen las mediciones solicitadas.
Esta tercera y última lista será la responsable de alimentar el proceso de transposición y de estandarización de valores.

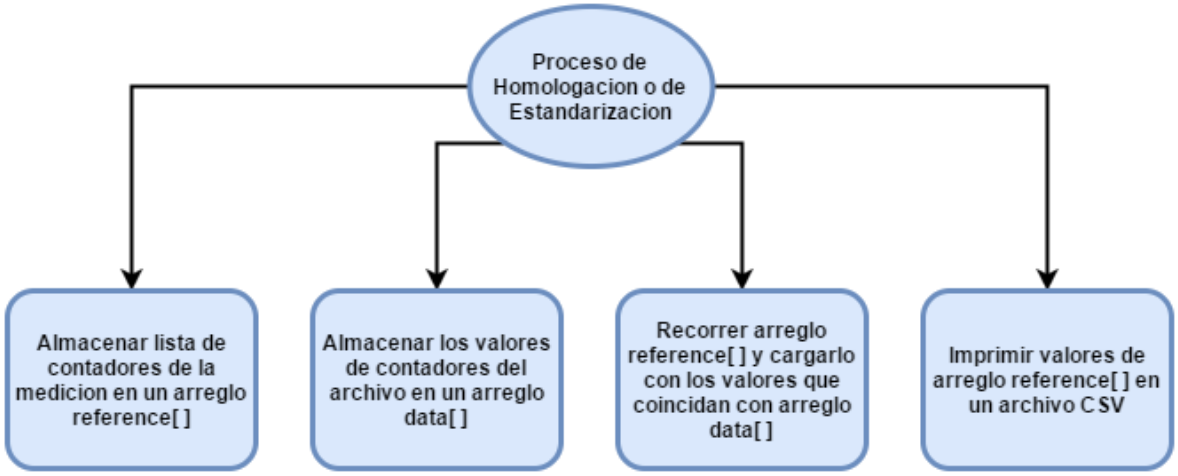
Diagrama de transposición de valores.



Grafica 3.3.3. Diagrama de descomposición funcional. Proceso de Transposición de valores.

El concepto de acoplamiento se puede ejemplificar en este proceso, porque todas las acciones de capturas son independientes unas de otras, no importa el orden ni el tamaño de los elementos, trabajan de manera individual.
De las cuatro acciones de captura de datos, dos son recursivas, esto quiere decir que se puede presentar más de un elemento dentro del XML que sean del mismo tipo y por tanto, sean necesarias las capturas de sus valores para conformar el valor necesitado.
Ante del ensamble final, se concatena los valores obtenidos de elemento del mismo tipo.
Recordemos que cada valor procesado se almacena en el formato "clave y valor".

Diagrama de Homologación/Estandarización de valores



Grafica 3.3.4. Diagrama de descomposición funcional. Proceso de Homologación de valores.

Junto con el proceso de transposición son los responsables del proceso de parseo del archivo XML guardando el resultado en un archivo CSV por medición, si es que un XML contiene más de una.
Este componente realiza tres acciones básicas para lograr el resultado deseado:

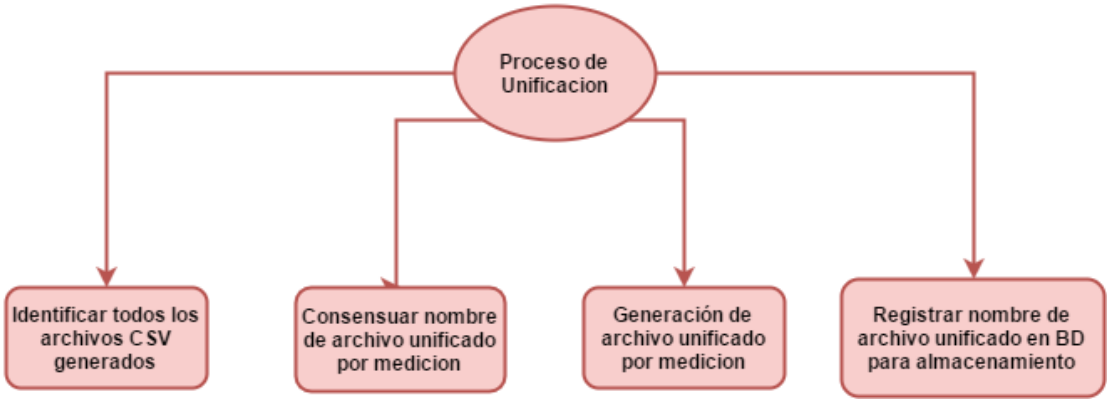
Primero, armamos un arreglo (array) en AWK con los nombres de los contadores que forman la medición, sin excepción, todos los contadores. Podemos llamar a este arreglo "reference" porque será nuestro parámetro de referencia siempre y de esta manera, aseguramos que la salida siempre será la misma.

Segundo, armamos un arreglo, al que llamaremos "data", con los datos que provienen del archivo saliente del proceso de transposición.
Recordemos que la salida estaba conformada por pares de datos que hacen referencia a un formato key-value (clave-valor).
Entonces podemos decir que siempre recibiremos dos valores por fila, el primero con el nombre del contador (la clave) y el segundo con el valor del mismo (el valor).

Y por último, hacemos un cruce de arreglos, entre "reference" y "data", teniendo como elemento en común el nombre de los contadores de la medición. Al final del cruce, el arreglo de referencia tendrá todos los valores de los contadores que se encuentren en el arreglo de datos.

Ampliaremos más este componente del proceso en el análisis de riesgos del proyecto.

Diagrama de unificación de valores

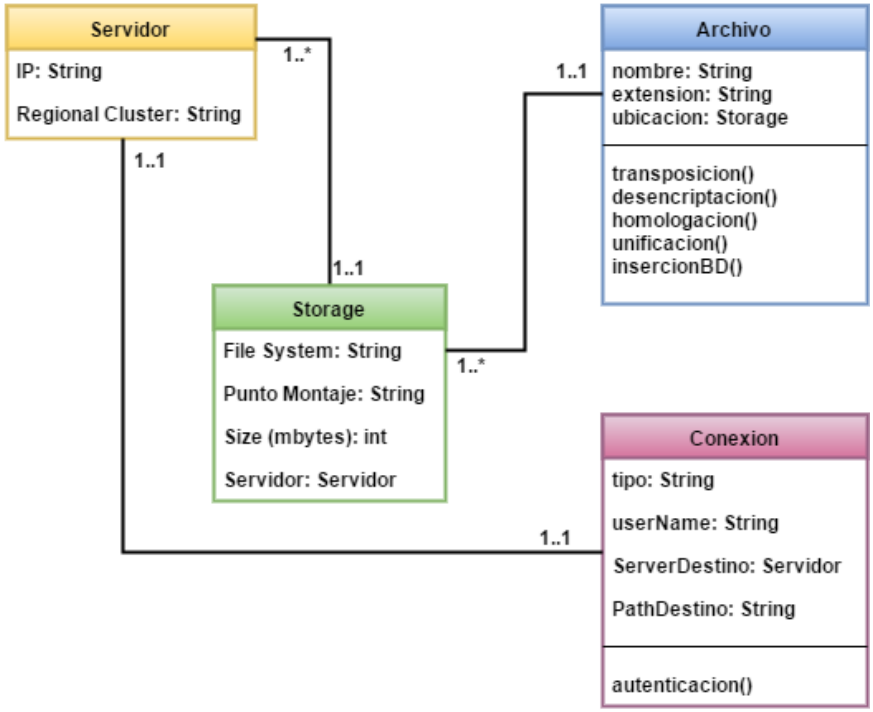


Grafica 3.3.5. Diagrama de descomposición funcional. Proceso de Unificación de Valores.

Es el último paso del proceso completo de parseo de archivos y más allá de que este último componente trabaje con el resultado del proceso de homologación, es parte vital para la inserción de los datos en la BD.
El objetivo es lograr mayor persistencia de los datos al almacenarlos en BD y esa acción de volcado de datos tiene que ser eficiente, es decir, que no se pierda demasiado tiempo ni utilice demasiados recursos de la BD.
Primero se identifican todos los archivos CSV generados para conocer cuáles son los elementos de red y mediciones que tenemos disponibles.
Con esos datos, construimos el nombre de un archivo que será el concentrador de toda la información por medición y tendrá por lo tanto, extensión ALL.
Luego, creamos el archivo ALL con el contenido estandarizado de cada archivo CSV previamente generado sin excluir o incluir, una sola línea, el resultado será un archivo concentrador con extensión ALL que contendrá el total de registros de todos los CSV por cada medición existente al momento.
Por último, este archivo se registra en una tabla de gestión dentro de la BD como “Pendiente” para que luego su contenido será almacenado en una tabla final.

Diagramas de Clases

Como consecuencia del punto anterior, el diagrama de clases del proceso general es la mejor explicación de cuáles son los principales elementos que componen el sistema, cuáles son sus relaciones y los posibles escenarios que se pueden presentar. A continuación, el diagrama de clases propuesto:



Grafica 3.3.6. Diagrama de Clases del proyecto.

Como clase principal, la clase "archivo" tiene atributos y métodos que son propios y únicos de la clase. Los atributos más relevantes son el nombre y extensión del archivo. Un dato a tener en cuenta es que conociendo la extensión del archivo podemos identificar en que parte del proceso de parseo se ubica ese archivo dentro de la carpeta destino.
Como un archivo no puede estar en más de un lugar físico, tiene una relación 1 a 1 con la clase "storage" pero si una clase "storage" puede tener muchos archivos.
Con respecto a los métodos, describiremos en una tabla cual es el tipo de archivo que entra a cada método y cuál es su respectiva salida.

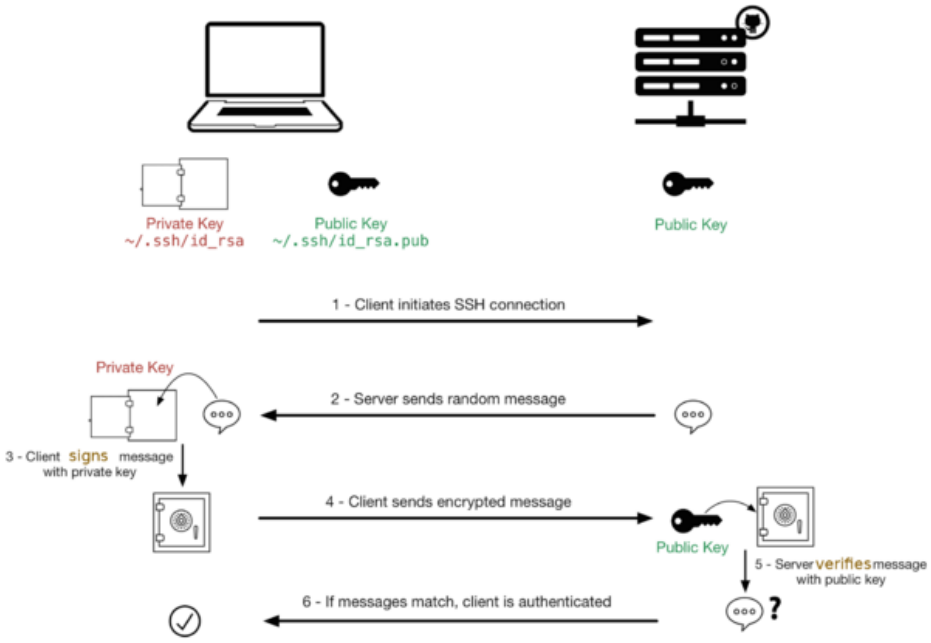
La clase "storage" tiene una relación 1 a 1 con la clase "servidor" pero la clase "servidor" puede tener muchas estar en muchas instancias de la clase "storage".
Y en la clase "conexión" se encuentra representada la relación entre el servidor origen (Regional Cluster) con el servidor destino (Internal Storage), uno de los atributos es el "tipo" de conexión que puede ser del tipo SSH, SCP o SFTP; todas son conexiones seguras y necesitan del método de autenticación para la construcción de ese vínculo de confianza entre ambos servidores, garantizando el establecimiento de conexiones seguras sin contraseña de forma automática.

Riesgos del proyecto

Analizaremos dos situaciones que se presentan en el proyecto y que son de conflicto porque condicionan la continuidad de la solución propuesta si el resultado no es eficiente.
A continuación, las situaciones:

- *Creación de conexiones entre los Regional Clusters y el Internal Storage que sean seguras, sin autenticación, de manera automática e invocables desde procesos Shell.*

La primera necesidad que se presenta es el tener que armar un escenario de confianza entre los distintos ambientes, necesario para el cruce de datos, sin que este afecte los procesos diseñados.
La solución tiene que ser transparente y garantizar disponibilidad, confianza y por sobre todo tiene que ser escalable, la misma solución se debe reutilizar en cualquier Regional Cluster.



Grafica 3.3.7. Diagrama de actividades. Proceso de establecimiento de conexión sin contraseña.

Los pasos para la concreción de esta solución son los siguientes:
Primero, se construyen un par de llaves necesarias para la conexión. Una es pública y la otra es privada.
Y segundo, le enviamos una copia de la llave pública al servidor remoto.

Recordemos que el objetivo es lograr una conexión segura y de confianza sin utilizar contraseñas.

Al momento de intentar establecer una conexión con un servidor, este nos envía un mensaje aleatorio. Nosotros ciframos el mensaje aleatorio recibido utilizando nuestra llave privada y se le enviamos nuevamente. El servidor utilizando la llave pública que recibió de nosotros con anterioridad, descifrara el mensaje y si coincide con el mensaje aleatorio que el envío, se establece la conexión segura sin necesidad de contraseñas.

- Garantizar un resultado estándar por tipo de medición sin importar el origen de los datos en el proceso de creación del archivo CSV, esto es el proceso de homologación/estandarización, usando AWK.

Esta situación es tan relevante o importante como la anterior. Que los archivos XML se puedan descomponer en elementos más pequeños con formato "key-value" no es resultado suficiente, porque el orden en que se presenten finalmente todos los elementos, es también una condición necesaria.

Usando las funciones para el manejo de arreglos disponible en AWK, se pueden resolver infinidad de escenarios, desde muy simples hasta muy complejos.
En nuestro caso y mirándolo desde una perspectiva de diseño, lo que se necesita es cruzar dos arreglos (uno de referencia y otro con datos) y que un arreglo se vaya completando con los valores del otro. Al final, tendríamos el array de referencia completo con los valores que se obtuvieron en el cruce, con el arreglo de datos.

A continuación, explicaremos brevemente cada escenario presentado en el proyecto y a su vez, cual es el valor deseado.

Escenario desordenado

Reference [ ]	
Clave	Valor
Counter001	NULL
Counter002	NULL
Counter003	NULL
Counter004	NULL
Counter005	NULL
Counter006	NULL
Counter007	NULL
Counter008	NULL
Counter009	NULL
Counter010	NULL

data [ ] desordenado	
Clave	Valor
Counter001	10
Counter003	30
Counter002	20
Counter004	40
Counter005	50
Counter007	70
Counter006	60
Counter008	80
Counter009	90
Counter010	100

Resultado	
Clave	Valor
Counter001	10
Counter002	20
Counter003	30
Counter004	40
Counter005	50
Counter006	60
Counter007	70
Counter008	80
Counter009	90
Counter010	100

Tabla 3.3.1. Ejemplo de situación con valores desordenados.

Los contadores que tiene el archivo es igual que el listado de referencia, simplemente que el orden no es mismo. Es preciso identificar cuáles son los elementos que no se encuentran en el lugar apropiado y ubicarlos.

Escenario incompleto

Reference [ ]	
Clave	Valor
Counter001	NULL
Counter002	NULL
Counter003	NULL
Counter004	NULL
Counter005	NULL
Counter006	NULL
Counter007	NULL
Counter008	NULL
Counter009	NULL
Counter010	NULL

data [ ] incompleto	
Clave	Valor
Counter001	10
Counter003	30
Counter004	40
Counter006	60
Counter007	70
Counter008	80
Counter010	100

Resultado	
Clave	Valor
Counter001	10
Counter002	NULL
Counter003	30
Counter004	40
Counter005	NULL
Counter006	60
Counter007	70
Counter008	80
Counter009	NULL
Counter010	100

Tabla 3.3.2. Ejemplo de situación con valores incompletos.

Puede suceder por distintas razones (versiones más nuevas de la medición que eliminan contadores obsoletos) que la lista de valores no complete todos los contadores de la medición y sino implementamos una solución el resultado final será más pequeño para algunos elementos y para otros no.
Es preciso identificar cuáles son los contadores pendientes y registrarlos de igual modo para estandarizar la salida.

Escenario con excedente de datos

Reference [ ]	
Clave	Valor
Counter001	NULL
Counter002	NULL
Counter003	NULL
Counter004	NULL
Counter005	NULL
Counter006	NULL
Counter007	NULL
Counter008	NULL
Counter009	NULL
Counter010	NULL

data [ ] excedido	
Clave	Valor
Counter001	10
Counter002	20
Counter003	30
Counter004	40
Counter005	50
Counter006	60
Counter007	70
Counter008	80
Counter009	90
Counter010	100
Counter011	110
Counter012	120

Resultado	
Clave	Valor
Counter001	10
Counter002	20
Counter003	30
Counter004	40
Counter005	50
Counter006	60
Counter007	70
Counter008	80
Counter009	90
Counter010	100

Tabla 3.3.3. Ejemplo de situación con valores de más.

En situación opuesta al anterior, la lista de valores puede ser más grande que la definida como referencia. En este caso, se eliminan los contadores que no figuran en la lista y se logra así, estandarizar la salida.

Escenario complejo

Reference [ ]	
Clave	Valor
Counter001	NULL
Counter002	NULL
Counter003	NULL
Counter004	NULL
Counter005	NULL
Counter006	NULL
Counter007	NULL
Counter008	NULL
Counter009	NULL
Counter010	NULL

data [ ] mix	
Clave	Valor
Counter002	20
Counter005	50
Counter003	30
Counter004	40
Counter009	90
Counter006	60
Counter007	70
Counter008	80
Counter020	200

Resultado	
Clave	Valor
Counter001	NULL
Counter002	20
Counter003	30
Counter004	40
Counter005	50
Counter006	60
Counter007	70
Counter008	80
Counter009	90
Counter010	NULL

Tabla 3.3.4. Ejemplo de situación de múltiples escenarios con valores desordenados, incompletos y de más.

Son todos los escenarios anteriores pero presentes en una lista de valores.
En este caso, el método de resolución es el mismo al de cualquier otro escenario y la salida es estándar por medición.

Cuarta Parte. Concreción del modelo.

Implementación

El código de proyecto es Script Shell y el kernel del proceso parser está desarrollado en AWK.
Partiendo de esta referencia comenzaremos a identificar cada uno de los componentes descriptos hasta el momento y como se materializaron en código dentro del proceso.

```
24  # Variables
25  logFile=nsnProcessFtpPMDataHourly.log
26  folder=`f_getDateHour_yyyyMMddHH 1`
27
28  cd $wDirProcessFtpPMDataHourly
29
30  > $logFile
31  f_log 'Start Main FTP Process PM Data Hourly' $logFile
32
33  for i in `echo $regionalClusterList`
```

Grafica 4.1.1. Construcción de la variable folder.

Dos variables se definen al comienzo del primer proceso, que es el proceso de recolección de datos: Una variable que identifica el nombre del archivo con el Log de proceso y la mas significa, la variable folder. La variable folder define la hora de las mediciones a recolectar. Es producto de una función f_getDateHour_yyyyMMddHH que devuelve el valor de la hora hacia atrás que se le pide en formato yyyymmddhh24.

```
/calidad/harriague/processFtp>#
/calidad/harriague/processFtp># date
Mon Nov 21 16:00:29 ART 2016
/calidad/harriague/processFtp># f_getDateHour_yyyyMMddHH 1
2016112115
/calidad/harriague/processFtp># f_getDateHour_yyyyMMddHH 2
2016112114
/calidad/harriague/processFtp>#
```

Grafica 4.1.2. Salida de la función f_getDateHour_yyyyMMddHH.

Esta y algunas otras funciones más de gestión se almacenan en el directorio de control, donde se guardan todas las funciones utilidades en el proyecto.
Aquí las sentencias de invocación a cada una de las bibliotecas para cargarlas en memoria y estén disponibles a lo largo del todo el proceso de parseo.

```
4  # Declaracion de Funciones
5  . /export/home/harriag/control/include.sh
6  . /export/home/harriag/control/unix.sh
7  . /export/home/harriag/control/variables.sh
8  . /export/home/harriag/control/loadProcess.sh
```

Grafica 4.1.3. Sentencias de invocación de funciones unix y programas AWK.

Como mencionamos, la variable folder define la hora de las mediciones a recolectar.

Pero todavía no sabemos a que regional cluster conectarnos, a continuación veremos ese punto.

```
33  for i in `echo $regionalClusterList`
34  do
35    f_log "FTP Process PM Data $i" $logFile
36
37    values=`f_GetValuesByRegionalCluster $i $folder`
38
39    flagPss=`echo $values | nawk '{ print $1 }'`
40    pathXML=`echo $values | nawk '{ print $3 }'`
41
42    userOssRc=`echo $values | nawk '{ print $4 }'`
43    nameOssRc=`echo $values | nawk '{ print $5 }'`
44    pathOssRc=`echo $values | nawk '{ print $6 }'`
45
46    if [ $flagPss -eq 1 ]; then
47    if [ -d $pathXML ]; then
48
49    f_log "Ya existe $pathXML" $logFile
50
51    else
52
53    scp -pr ${userOssRc}@${nameOssRc}:${pathOssRc} $pathXML
54    gzip -drf $pathXML
55    chmod 777 -Rf $pathXML
56
57    f_log "Execution Call Counter Value $i" $logFile
58    callConstructorCounterValue.sh $i $folder
59
60    fi
61    fi
62  done
```

Grafica 4.1.4. Bucle del tipo FOR, con sentencias que se ejecutaran por cada regional.

Lo que sigue es una estructura de tipo bucle que contiene las sentencias que se ejecutaran por cada regional.

En la línea 33, por cada valor contenido en la variable regionalClusterList, se almacena en la variable i.
En la línea 37, ahora si junto con la variable folder que tiene la hora de las mediciones a recolectar, obtendremos todos los valores que se necesitan para iniciar el proceso de conexión y de extracción de los datos.

Y ahora si, a partir de este momento podemos decir que comenzamos a hacer referencias a los distintos elementos antes descriptos.

En la línea 40, la variable pathXml contendría la ubicación de los archivos XML luego de ser extraídos del regional correspondiente. Esta variable hace referencia al parámetro "ubicación" de la clase archivo.

En la línea 43, hacemos referencia a la clase Servidor. Es el nombre del servidor Remoto y es utilizada en la clase de conexión.

En la línea 57, hacemos referencia a la clase de conexión. Los elementos que componían la clase eran: el tipo de conexión (scp), el userName (userOssRc), el serverDestino (nameOssRc) y el pathDestino (pathOssRc).

Arquitectura del Proceso

Se adjunta en el anexo del documento de grado, un esquema de la arquitectura completa, de todas las funciones utilizadas y en el orden en que son invocadas.

Estructuración de los componentes

Paths

Bibliotecas	/export/home/harriag/control/
Home	/calidad/harriague/processFtp/
Lista de Mediciones	/calidad/harriague/processFtp/listMeasurementUnique/

Tabla 4.1.1. Esquema de directorios.

Todos los elementos que componen el proceso parseador, se encuentran en estas tres rutas.
La primera, hace referencia a la ruta en donde se almacenan todas las funciones UNIX diseñadas para ser utilizadas en el proceso.
Por su semejanza conceptual en el guardado e invocación de funciones con el del lenguaje C, le hemos puesto "biblioteca" a ese lugar de referencia en donde se guardan las funciones, pero todas las funciones se encuentran dentro archivos SH, es decir, de tipo Script Shell.

La segunda, es la ruta donde están todos los elementos y componentes principales.

Por ultimo, la tercera es la ruta en donde están las listas de todas las mediciones con sus contadores. Las listas se utilizan en el proceso de estandarización de la salida del archivo XML.

Storage

Storage		
OSSRC	Path External	Path Internal
OSSRC1	/var/opt/nokia/oss/global/mediation/north/pm/export/	/calidad/data/nsn/storage/xml/rc1/pm/
OSSRC2	/var/opt/nokia/oss/global/mediation/north/pm/export/	/calidad/data/nsn/storage/xml/rc2/pm/
OSSRC3	/var/opt/nokia/oss/global/mediation/north/pm/export/	/calidad/data/nsn/storage/xml/rc3/pm/

Tabla 4.1.2. Relación de directorios, regional por regional, con los archivos XML dentro del Regional Cluster y su ubicación destino dentro del Storage de la jefatura.

Estas son las rutas tanto del External Storage como del Internal Storage, discriminadas por Regional Cluster.
Basicamente, la ruta external es la misma siempre sin importar el regional que se trate.
En cambio, la ruta internal al ser dentro de un mismo equipo, se discrimina por carpetas.

Componentes y Funciones

Elemento	Orden Ejecucion	Componente	Tipo	Ubicación	Descripcion
1	1	nsnProcessFtpPMDataHourly.sh	Shell Scripting	Home	Proceso de conexión y de recoleccion de archivos
2	2	callConstructorCounterValue.sh	Shell Scripting	Home	Invocador de funciones del proceso parseador
3	2.1	f_constructorListXmltoParser	Shell Function	Biblioteca loadProcess.sh	Constructor de la lista de archivos a procesar
4	2.2	f_callConstructorCounterValue	Shell Function	Biblioteca loadProcess.sh	Invocador de lista de mediciones y del proceso parseador
5	2.2.1	f_constructorCounterValue	Shell Function	Biblioteca loadProcess.sh	Proceso Parseador
6	2.2.1.1	constructorCounterValue.firstPart.awk	AWK Program	Home	Proceso de transposicion de valores
7	2.2.1.2	constructorCounterValue.secondPart.awk	AWK Program	Home	Proceso de estandarizacion de valores
8	2.3	f_constructorOneCsvFileByPathByNeByMea2	Shell Function	Biblioteca loadProcess.sh	Proceso de unificacion de archivos
9	3	constructorFileExternal.sh	Shell Scripting	Home	Proceso de insercion de lista de archivos en BD

Tabla 4.1.3. Orden de ejecución, nombre de los componentes de la solución y su ubicación dentro del esquema de directorios.

Este es el listado de los componentes y es la mejor aproximación a una relación entre el concepto de proceso y su componente físico.
Primero, el elemento nro.1 es el responsable de la conexión y de la recolección de los datos.
Segundo, el elemento nro.2 es el responsable de ejecutar las funciones que llevan a cabo las acciones 2.1, 2.2 y 2.3.
El elemento 5 es una función que concentra los elementos 6 y 7. Entre ambos se completa el proceso de parseo.
Y por último, el elemento 9 es el responsable de listar los archivos ALL generados en el elemento 8 e insertar los nombres de los archivos en la BD para su posterior almacenamiento.

Puesta en Marcha

Para la puesta en marcha se necesita la definición de una ventana de mantenimiento de 15 minutos aproximadamente, que solamente se utiliza para deshabilitar el ingreso de datos desde la BD y habilitar el ingreso de datos desde los archivos ALL.
La solución es transparente y se pensó en que la puesta en marcha sea lo menos invasiva posible.
Un punto a considerar es que no hay rollback de la implementación, porque el modelo de captura de datos desde BD esta caducado y es necesario reemplazarlo. No se puede evaluar un rollback de la solución, caso contrario se verá la implementación de alguna solución workaround para evitar conflictos.

Las capacitaciones y las guías de resolución de problemas que se dictaron, se podrán ver en el anexo, son imágenes de lo que se dio en las capacitaciones al proceso como así también, un esquema de resolución de conflictos o de problemas.

Conclusiones

Se respira un aire de conformidad en la solución y se fue viviendo a medida que se cumplían los plazos de las entregas parciales y los temas pendientes se iban uno a uno cerrando, concretando. Las expectativas eran grandes y se cubrían exitosamente.

Desde la construcción del ambiente de confianza para la autenticación sin contraseña hasta el proceso de unificación de archivos ya parseados, todos fueron logros producto de un tiempo de estudio, de dedicación y de constancia en el uso de las herramientas utilizadas.

Con respecto a los objetivos propuestos:

- Ante el escenario inminente del cierre de las BDs de Nokia como proveedor de la información de performance, el objetivo de lograr continuidad en la recepción de los datos trayéndolos ahora desde un repositorio externo con cientos de XML, todo esto en un plazo de entrega mínimo y haciéndolo de manera eficiente, se pudo lograr.
- Otro objetivo importante fue el construir una solución escalable, reutilizable para cualquier tipo de medición, cualquier tipo de elemento de red y a su vez fácil de mantener, permitiendo la posibilidad de tener nueva información parseada de una manera ágil, fácil y sencilla.

Los resultados obtenidos fueron muy alentadores y aseguran de alguna manera, la sustentabilidad de la solución y el visto bueno de la jefatura en mantenerla.

En lo personal, estoy muy conforme con la solución por el alto grado de acoplamiento que hace sencillo su mantenimiento y que sienta las bases para pensar en escalar.
Fue un logro personal también, siento que todos los años como desarrollador Unix se plasmaron en una solución que es aplaudida por los superiores y eso me lleno de orgullo.

Finalizando, concluimos que todo lo planeado y lo definido al comienzo del proyecto, se cumplió y se alcanzaron todas las metas establecidas en los plazos acordados y de manera eficiente, porque el margen de error fue mínimo.

Bibliografía

- Nombre Sitio: The Geek Stuff | Titulo Articulo: *AWK Arrays Explained with 5 Practical Examples* | Fecha Publicación: 10 de Marzo de 2010 por Ramesh Natarajan | Dirección Electrónica: http://www.thegeekstuff.com/2010/03/awk-arrays-explained-with-5-practical-examples/

- Nombre Sitio: Tapping Away | Titulo Articulo: *Awking it – how to load a file into an array in awk* | Fecha Publicación: 18 de Marzo de 2011 | Dirección Electrónica: https://magvar.wordpress.com/2011/05/18/awking-it-how-to-load-a-file-into-an-array-in-awk/

- Nombre Sitio: RICH And ATLAS Computing Facility at Rrookhaven National Laboratory | Titulo Articulo: *UNIX SSH Key Generation* | Fecha Publicación: 27 de Octubre 2016 | Dirección Electrónica: https://www.racf.bnl.gov/docs/authentication/ssh/sshkeygenunix

- Nombre Sitio: TECMINT: LINUX HOWTOS, TUTORIALS & GUIDES | Titulo Articulo: *SSH Passwordless Login Using SSH Keygen in 5 Easy Steps* | Fecha Publicación: 9 de Abril de 2015 por Ravi Saive | Dirección Electrónica: http://www.tecmint.com/ssh-passwordless-login-using-ssh-keygen-in-5-easy-steps/

- Nombre Sitio: Tutorials Point. Simply Easy Learning | Titulo Articulo: *Unix - Shell Functions* | Direccion Electronica: http://www.tutorialspoint.com/unix/unix-shell-functions.htm

- Nombre Sitio: PowerData. Especialistas en Gestión de Datos | Titulo Articulo*: Procesos ETL: Definición, Características, Beneficios y Retos* | Fecha de Publicación: 9 de Julio de 2013 | Dirección Electrónica: http://blog.powerdata.es/el-valor-de-la-gestion-de-datos/bid/312584/Procesos-ETL-Definici-n-Caracter-sticas-Beneficios-y-Retos

- Nombre Sitio: w3schools.com THE WORLD'S LARGEST WEB DEVELOPER SITE | Titulo Articulo: *XML Tutorial* | Direccion Electronica: http://www.w3schools.com/xml/

- Nombre Sitio: #awk | Titulo Articulo: *XMLScraping* | Fecha Publicación: 23 de Noviembre de 2011 por Pierre Gaston | Dirección Electrónica: http://awk.freeshell.org/XMLScraping

- Nombre Sitio: Odd Bits | Titulo Articulo: *Parsing XML with Awk* | Fecha Publicación: 10 de Setiembre de 2012 por Lars Kellogg-Stedman | Dirección Electrónica: http://blog.oddbit.com/2012/09/10/awk-parsing-xml/

- Nombre Sitio: A Bash Shell Scripting Directory. For Linux and UNIX-like systems, organized by topic into categories | Titulo Articulo: *Linux Shell Scripting Tutorial (LSST) v2.0* | Fecha Publicación: 5 de Febrero de 2011 por Vivek Gite | Dirección Electrónica: https://bash.cyberciti.biz/guide/Main_Page

- Nombre Sitio: AWK Language Programming. A User's Guide for GNU AWK | Titulo Articulo: *Arrays in awk* | Fecha Publicación: Enero del 1996 por Arnold D. Robbins | Dirección Electrónica: https://www.chemie.fu-berlin.de/chemnet/use/info/gawk/gawk_toc.html

Glosario

Abreviatura	Significado
ETL	**Extract, Transform and Load** («extraer, transformar y cargar», frecuentemente abreviado **ETL**) es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos, data mart, o data warehouse para analizar, o en otro sistema operacional para apoyar un proceso de negocio.
XML	Siglas en inglés de *eXtensible Markup Language*, traducido como "Lenguaje de Marcado Extensible" o "Lenguaje de Marcas Extensible", es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible
BD	Base de Datos
AWK	Es un lenguaje de programación diseñado para procesar datos basados en texto, ya sean ficheros o flujos de datos. El nombre **AWK** deriva de las iniciales de los apellidos de sus autores: Alfred Aho, Peter Weinberger, y Brian Kernighan.
GSM	El **sistema global para las comunicaciones móviles** (del inglés *Global System for Mobile communications*, **GSM**, y originariamente del francés *groupe spécial mobile*) es un sistema estándar, libre de regalías, de telefonía móvil digital. GSM se considera, por su velocidad de transmisión y otras características, un estándar de segunda generación (2G).
UMTS	**Sistema universal de telecomunicaciones móviles** (***Universal Mobile Telecommunications System*** o ***UMTS***) es una de las tecnologías usadas por los móviles de tercera generación, sucesora de GSM, debido a que la tecnología GSM propiamente dicha no podía evolucionar para prestar servicios considerados de tercera generación.
LTE	Long Term Evolution (LTE, por sus siglas en inglés, lo que en español se traduce como Evolución a largo plazo), en telecomunicaciones, es un estándar para comunicaciones inalámbricas de transmisión de datos de alta velocidad para teléfonos móviles y terminales de datos. Es un protocolo de la norma 3GPP definida por unos como una evolución de la norma 3GPP UMTS (3G), y por otros como un nuevo concepto de arquitectura evolutiva (4G).
NSN	Nokia Siemens Network
BTS	Antena GSM
WCELL	Antena UMTS
LNCEL	Antena LTE
DBLINK	Data Base Link o conexión entre base de datos
RNC	Una RNC (del inglés **Radio Network Controller**, Controlador de la Red Radio) es un elemento de red de alta jerarquía de la red de acceso de la tecnología UMTS, responsable del control de los nodos b que se conectan a ella. La RNC se encarga de la gestión de recursos radio (RRM) y parte de la gestión de movilidad (MM). Además es el punto en el que encriptación de los datos, antes de que sean enviados desde o hacia el

	terminal móvil. La RNC se conecta a la red de núcleo de conmutación de circuitos (CS-CN, *Circuit Switched Core Network*) a través del *Media Gateway* (MGW) y del SGSN (*Serving GPRS Support Node*) en la red de núcleo de conmutación de paquetes (PS-CN, *Packet Switched Core Network*).
BSC	El empleo de celdas requiere de una capa adicional de red que es novedosa en el estándar GSM respecto a los sistemas anteriores: es el **controlador de estaciones base, o BSC,** *(Base Station Controller)* que actúa de intermediario entre el "corazón" de la red y las antenas, y se encarga del reparto de frecuencias y el control de potencia de terminales y estaciones base.
POC	Una **prueba de concepto** o PoC (por sus siglas en inglés) es una implementación, a menudo resumida o incompleta, de un método o de una idea, realizada con el propósito de verificar que el concepto o teoría en cuestión es susceptible de ser explotada de una manera útil.
CSV	Los archivos **CSV** (del inglés *comma-separated values*) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el Separador decimal: Argentina, España, Brasil...) y las filas por saltos de línea.
Sql Loader	Sql * Loader es la herramienta de carga de tablas de Oracle. En la terminología de bases de datos se entiende que *cargar una tabla* es incorporar datos a la misma.
Sql Plus	**SQL*Plus** es un programa de línea de comandos de Oracle que puede ejecutar comandos SQL y PL/SQL de forma interactiva o mediante un script.
OMeS	Open Measurement Standard
SSH	**SSH** (**S**ecure **SH**ell, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red.
VM	Virtual Machine
RC	Regional Cluster

Anexo

Construcción del escenario de confianza para las conexiones SSH sin contraseña

Veremos como un cliente se puede conectar a un servidor SSH, sin la necesidad de estar introduciendo la contraseña cada vez que nos queremos conectar al servidor. La forma de conseguir esto será mediante el uso de un par claves asimétricas.

El hecho de conectarse a un servidor SSH de forma segura, y sin la necesidad de introducir contraseña, tiene numerosas ventajas. Algunas de las ventajas que tendremos son las siguientes:

1. No tendremos que estar recordando e introduciendo contraseñas para conectarnos a un servidor SSH. Además al no introducir nuestra contraseña al autentificarnos, evitaremos que nuestra contraseña pueda ser robada mediante un ataque "man in the middle".
2. Es sumamente útil para automatizar tareas de copias de seguridad dentro o fuera nuestra red local. El envío del contenido se haría de forma cifrada por medio de SSH, y el proceso seria completamente automático y no tendríamos que introducir ninguna contraseña.
3. Para automatizar cualquier tipo de conexión remota y de tarea entre un servidor y un cliente. Como ejemplo a lo que acabamos de citar podemos leer el punto 2 de este apartado.

   Nota: Hoy en día prácticamente el 100% de distribuciones Linux disponen de un servidor SSH instalado de serie. Por lo tanto casi nadie lo debería instalar, caso contrario, sería necesario instalarlo.

Crear los pares de llaves Asimétricas

Una vez estamos seguros que el servidor SSH y el cliente tenga los paquetes necesarios, ya podemos generar las claves asimétricas para poder acceder a nuestro servidor SSH sin necesidad de introducir ninguna contraseña.
Para ello en el ordenador que actuará como cliente tenemos que abrir una terminal y teclear el siguiente comando:

```
ssh-keygen -b 4096 -t rsa
```

El significado de cada unos de los parámetros del comando es el siguiente:

ssh-keygen : Es el comando que genera el par de claves.

-b 4096 : Estamos indicando que la clave asimétrica que se generará tenga un tamaño de 4096 bits. Otros tamaños que podemos elegir por ejemplo son 1024 o 2048.

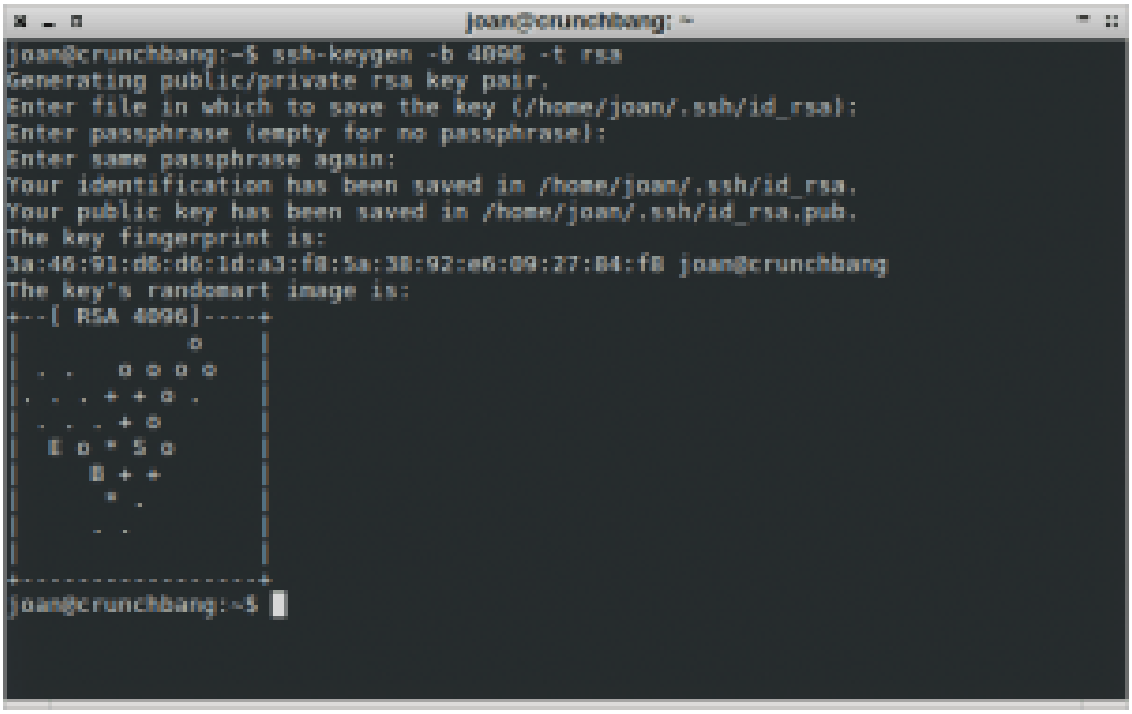
-t rsa : Indica que el algoritmo usado para generar el par de claves tiene que ser el rsa. Otros algoritmos que podemos usar son el dsa, ecdsa, rsa1 y ed25519.

Justo después de ejecutar el comando, se nos preguntará la ubicación donde queremos guardar las claves y el nombre que les queremos poner.
Cuando nos haga esta pregunta presionaremos la tecla **Enter**. De este modo las claves que generaremos se guardaran en la ubicación estandar que es la **/home/usuario/.ssh/**, y tendrán el nombre estandar que es **id_rsa**.

Seguidamente se nos preguntará si queremos introducir una contraseña para cifrar nuestra clave privada. Como queremos conectarnos al servidor sin necesidad de introducir ninguna contraseña, presionamos la tecla **Enter** sin introducir ninguna contraseña.
Finalmente se nos pregunta que volvamos a introducir la contraseña que acabamos de introducir. Como no hemos introducido ninguna contraseña volvemos a presionar la tecla **Enter**.

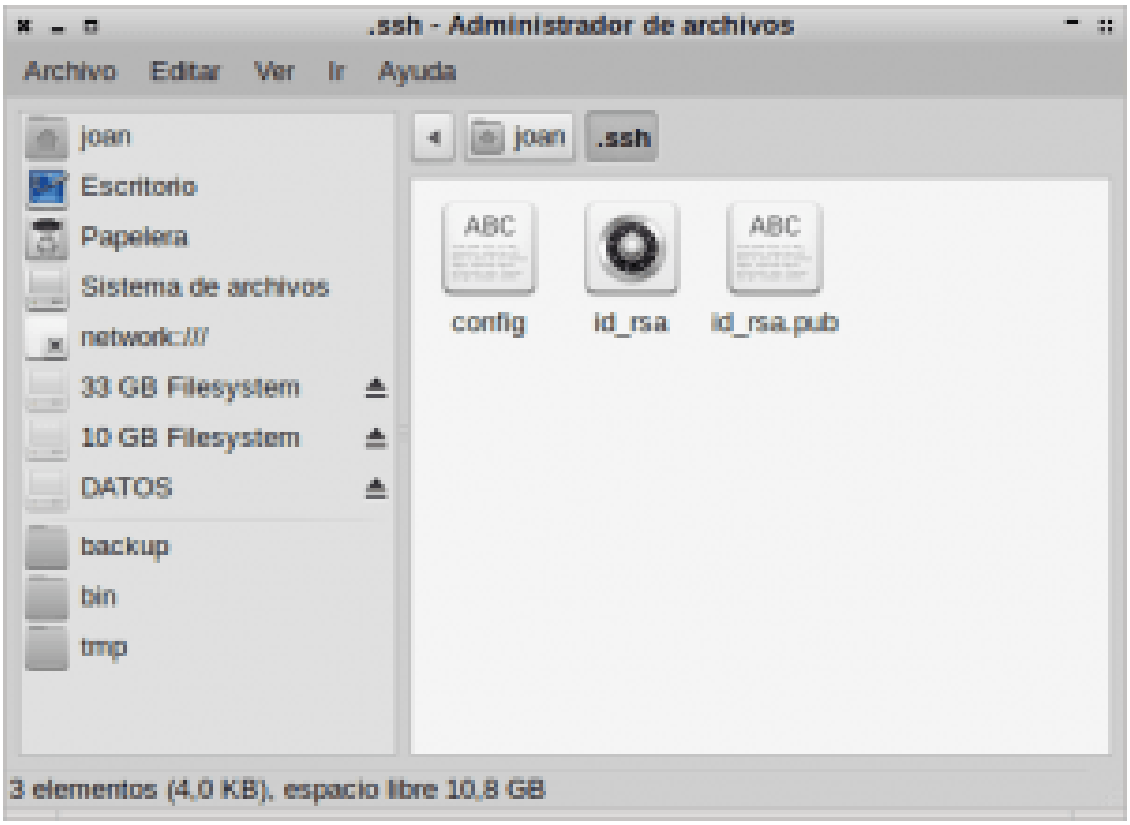
Nota: Al no introducir un password, la clave privada se guarda en nuestro disco duro sin cifrar. Esto veremos más adelante que supone un riesgo de seguridad.

Después de realizar estos pasos se crearan las claves asimétricas en la ubicación **~/.ssh**. En la siguiente captura de pantalla se puede ver un resumen de todos los pasos realizados.



Grafica A.1.1 Generación de Claves Asimétricas. La Pública y la Privada.

Si queremos, tal y como se puede ver en la captura de pantalla, podemos ir a la ubicación **~/.ssh** y comprobar que efectivamente se ha creado una clave privada con nombre **id_rsa**, y una clave pública con nombre **id_rsa.pub** que es la que deberemos exportar al servidor ssh.



Grafica A.1.2. Visualización correcta de claves en .ssh folder.

Agregar la llave publica del cliente en el Servidor

El siguiente paso es exportar la clave pública del ordenador que actúa como cliente, al ordenador que actúa como servidor.
Para ello tenemos que realizar los siguientes pasos.

Crear el archivo donde se guardaran las claves públicas autorizadas (acción en el servidor)

En el servidor SSH abrimos una terminal. Una vez abierta la terminal accedemos al directorio **~/.ssh** con el siguiente comando:

```
cd ~/.ssh
```

Una vez dentro del directorio **~/.ssh**, vamos a crear un archivo donde se van a guardar las claves públicas que nuestro servidor aceptará. Para ello tecleamos el siguiente comando en la terminal:

```
touch authorized_keys
```

Una vez tecleado el comando se creará el archivo **authorized_keys** en la ubicación **~/.ssh**, y todos los pasos a realizar en el servidor habrán terminado. En la siguiente captura de pantalla se pueden ver los pasos realizados hasta el momento en el servidor:

Grafica A.1.3. En el Servidor, creando el archivo authorized_keys.

Copiar la clave pública del cliente al servidor SSH (acción en el cliente)

Una vez terminado con el servidor, ahora tenemos que trabajar en el ordenador que actuará como cliente.

En el ordenador cliente abrimos una terminal. Una vez abierta la terminal accedemos al directorio **~/.ssh**, que es la que contiene nuestra clave pública, con el siguiente comando:

```
cd ~/.ssh
```

Una vez dentro de la carpeta **~/.ssh** exportaremos la clave pública al servidor usando el siguiente comando:

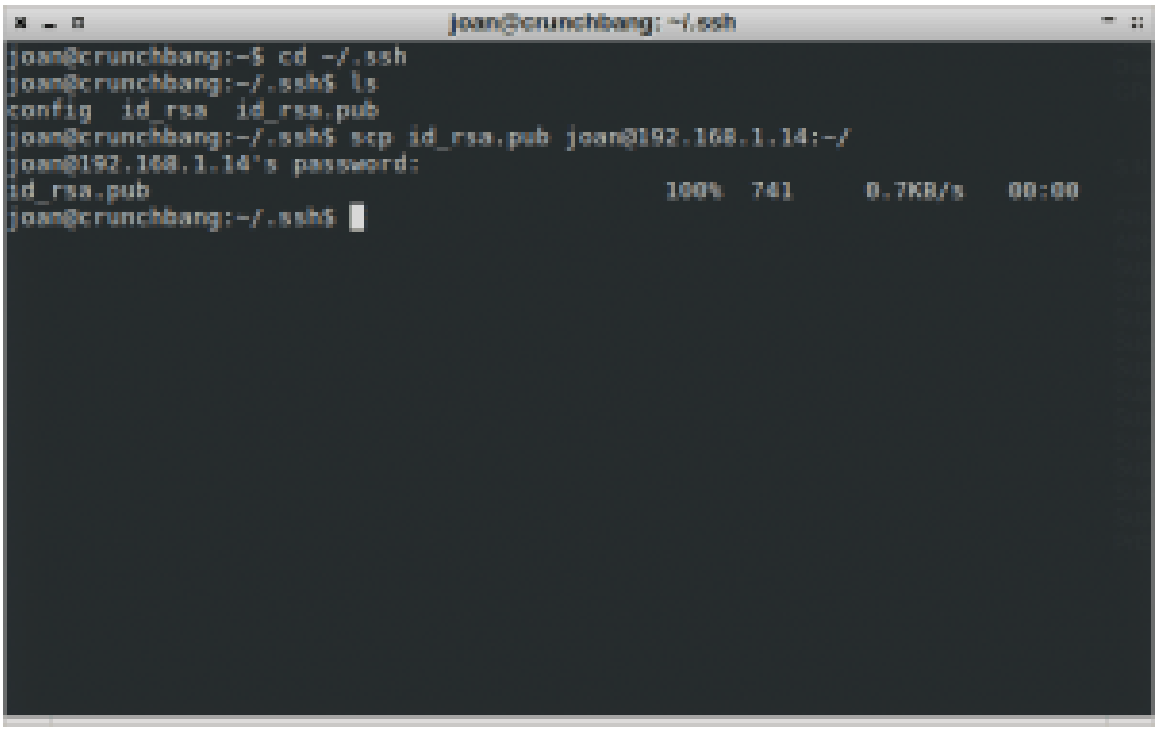
```
scp id_rsa.pub joan@192.168.1.14:~/
```

El significado de cada unos de los parámetros del comando es el siguiente:

scp : Es el comando scp (secure copy) sirve para copiar archivos de forma cifrada entre un sistema local y un servidor remoto.
id_rsa.pub : Es el nombre de la clave pública que queremos copiar al servidor.
joan@192.168.1.14:~/ : Es la dirección del servidor donde queremos copiar la clave pública. El usuario del servidor es joan, la ip del servidor es 192.168.1.14 y queremos que la clave pública se guarde en la ubicación home (**~/**).

Nota: Obviamente los comandos de este apartado hay que adaptarlos en función de las características de vuestra red local o de vuestra red externa.

Una vez se haya ejecutado el comando, se nos preguntará la contraseña del servidor SSH al que queremos copiar la clave pública. Introducimos la contraseña y justo después de introducirla, la clave pública del cliente se copiara la ubicación home (**~/**) del servidor SSH. Seguidamente se muestra una captura de pantalla de los pasos realizados en este apartado:



Grafica A.1.4. Copiando la clave publica en el Servidor.

Autorizar la clave pública del cliente en el servidor (acción en el servidor)

Una vez tenemos la clave pública del cliente en la home del servidor, tan solo nos queda introducir la clave pública del cliente a la lista de claves autorizadas del servidor **authorized_keys**, que creamos en el inicio del paso 4. Para ello seguimos los siguientes pasos.
En el servidor tenemos que ir a la ubicación donde hemos copiado la clave pública del cliente. Para ello en la terminal ejecutamos el siguiente comando:

```
cd ~/
```

Una vez ubicados en la home, que es donde copiamos la clave pública, ejecutamos el siguiente comando para añadir la clave pública en el fichero de autorización de claves públicas **authorized_keys**:

```
cat id_rsa.pub >> ~/.ssh/authorized_keys
```

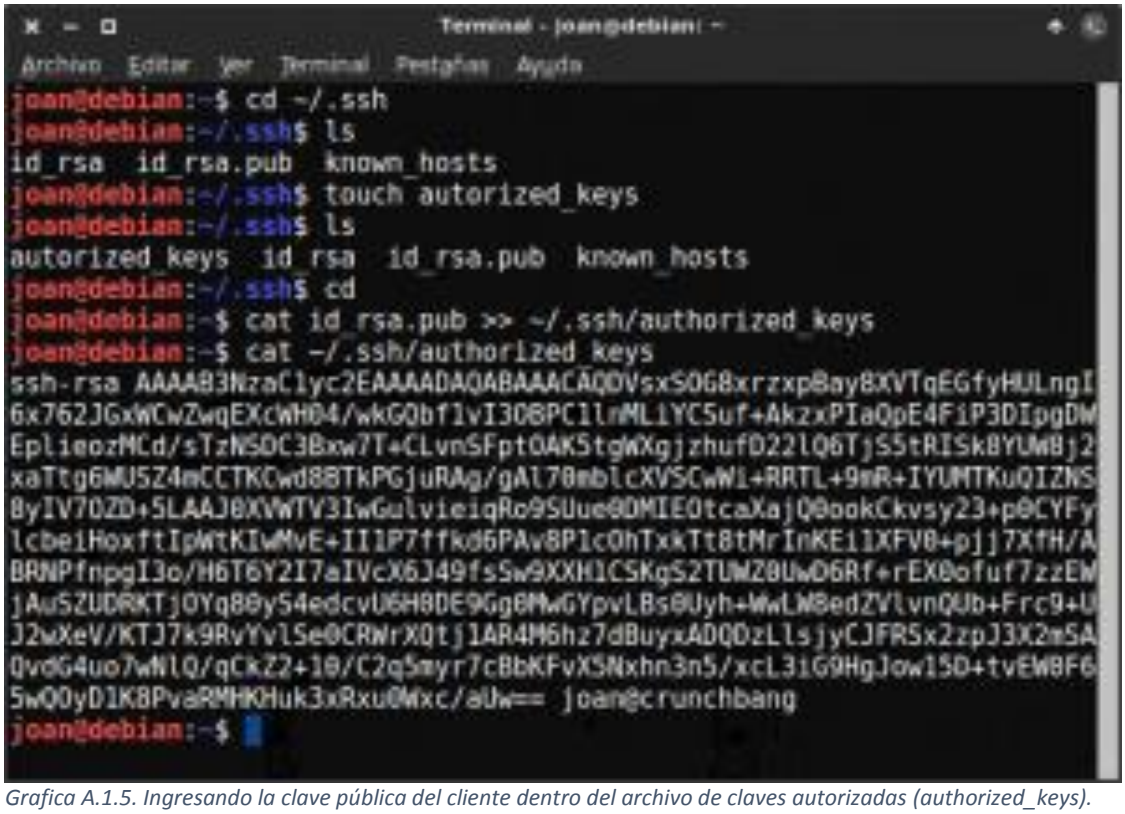
El significado de cada unos de los parámetros del comando es el siguiente:

cat: El Comando cat lo usaremos para añadir la clave pública dentro del fichero authorized_keys.
id_rsa.pub : Es el nombre de la clave pública que queremos autorizar.
>> ~/.ssh/authorized_keys : Es la ruta del archivo que almacenará las claves públicas autorizadas. La parte del comando >> es importante ya que es la parte posibilita que se añada la clave pública al fichero authorized_keys.

Una vez ejecutado el comando, la clave pública se copiará al fichero de claves autorizadas. Para comprobar que es que así podemos usar el siguiente comando:

```
cat ~/.ssh/authorized_keys
```

Si después de ejecutar el comando nos aparece una clave en pantalla, el proceso ha sido un éxito y ya nos podremos conectar al servidor SSH sin necesidad de introducir ninguna contraseña. En la siguiente captura de pantalla se muestran los pasos que se han seguido para autorizar la clave pública:



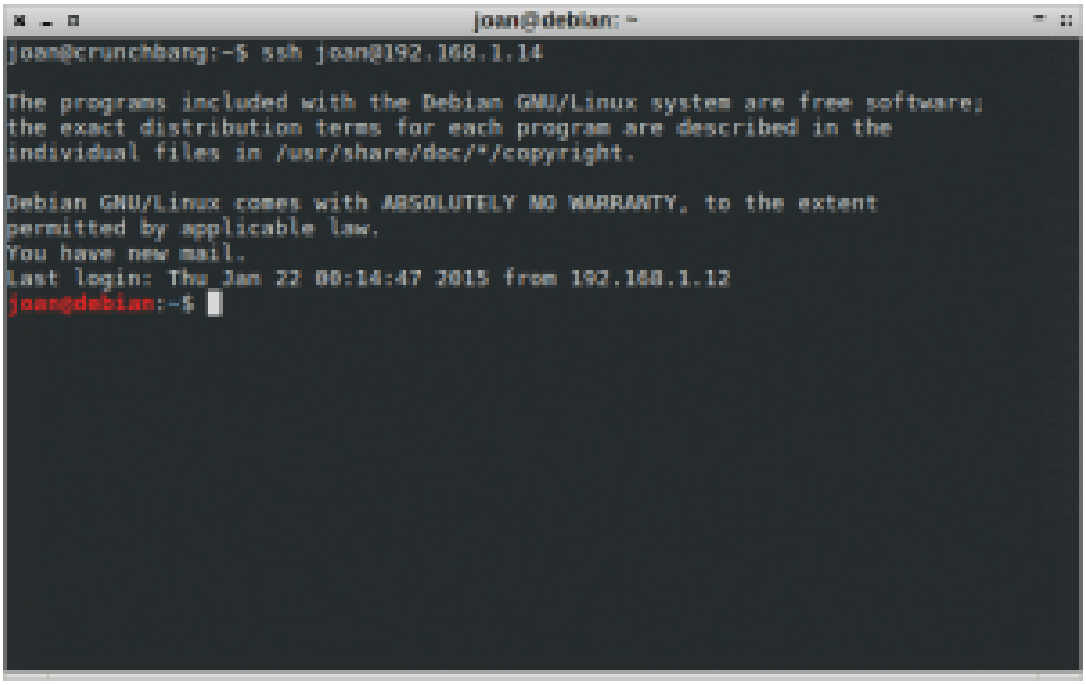
Grafica A.1.5. Ingresando la clave pública del cliente dentro del archivo de claves autorizadas (authorized_keys).

Nota: Una vez autorizada la clave pública, si queremos podemos borrar el archivo id_rsa.pub ubicado en la home del servidor.

Comprobación del Funcionamiento

Para comprobar el funcionamiento tan solo tenemos que intentar acceder a nuestro servidor SSH, tal y como lo hacemos de forma habitual. Si lo hacemos, tal y como se puede ver en la

captura de pantalla de este apartado, veremos que no se nos pide ninguna contraseña para acceder al servidor SSH.



Grafica A.1.6. Comprobacion del éxito en la creación del escenario de confianza.

Funcionamiento del Proceso de Autenticación

Una vez implementados los pasos citados, es interesante comentar cómo funciona el proceso de autentificación entre cliente y servidor.

El procedimiento de autentificación a grandes rasgos funciona de la siguiente forma:

1. Cuando el cliente intenta acceder al servidor SSH. Lo primero que hace el servidor es comprobar si la clave pública del cliente está autorizada. Si está autorizada el proceso de autentificación sigue adelante. Si no está autorizada el proceso termina y no podemos acceder al servidor.
2. Si la clave pública del cliente está autorizada por el servidor, el servidor cifra un mensaje con la clave pública del cliente. Una vez el servidor ha cifrado el mensaje lo envía al cliente.
3. El cliente recibe el mensaje del servidor. Una vez recibido el mensaje, el cliente intenta descifrar este mensaje con la clave privada. Si usando la clave privada el cliente descifra el mensaje, el servidor lo detectará y se establecerá la conexión con el servidor SSH. Si el cliente no puede descifrar el mensaje enviado por el servidor, se abortará el proceso de conexión al servidor.

Cuestiones de seguridad a tener en cuenta

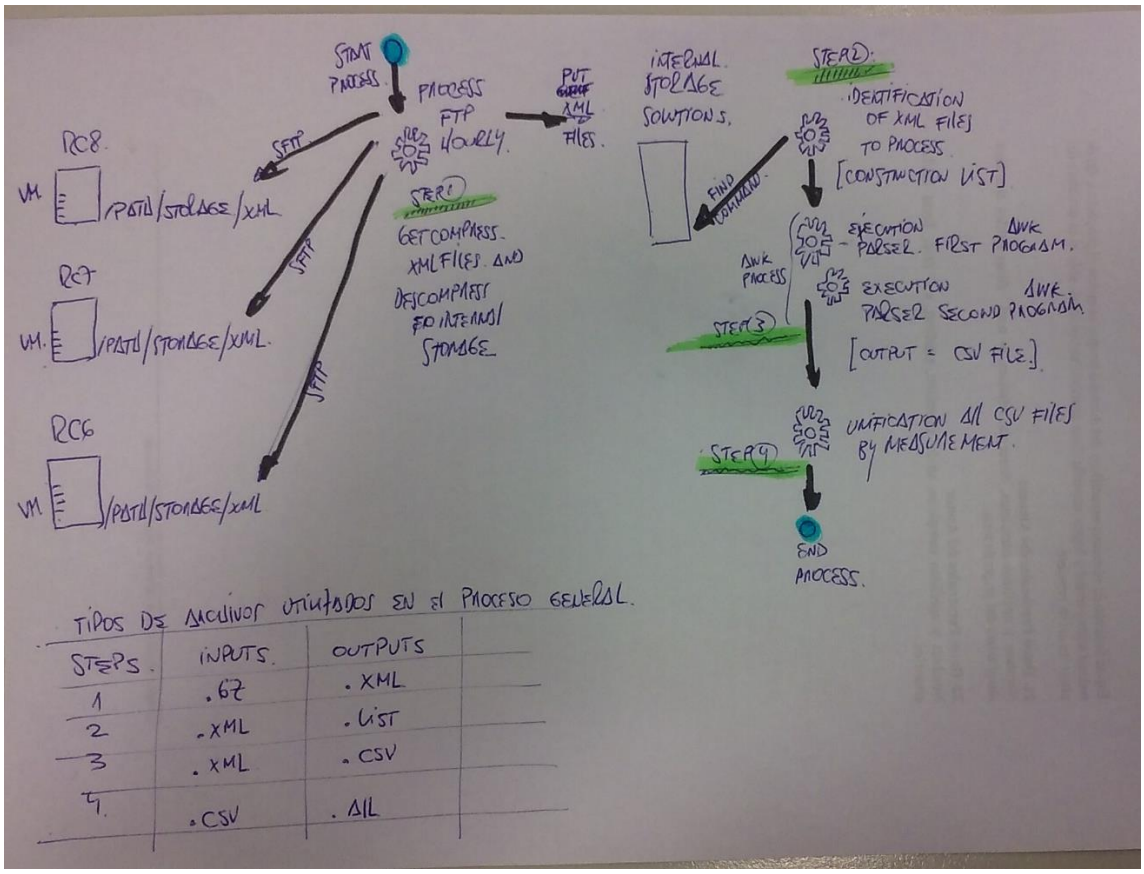
Si analizamos el proceso de autentificación del apartado anterior vemos que existe un riesgo importante. Cualquier persona que se apodere de nuestra clave pública y privada, es

muy posible que tenga la posibilidad de acceder a nuestro servidor SSH. Sin duda esto es un riesgo en lo que a seguridad se refiere.

La solución al problema que acabamos de describir es relativamente sencilla. Lo que deberíamos realizar es introducir una contraseña a nuestra clave privada. De este modo cada vez que quisiéramos entrar en nuestro servidor SSH tendríamos que añadir una contraseña, y aunque alguien pudiera obtener nuestro par de claves no se podría autentificar porque no sabría la contraseña. Si aplicamos lo que acabo de comentar, para intentar evitar estar poniendo la contraseña constantemente, deberíamos usar ssh-agent para que únicamente tengamos que introducir la contraseña de la clave privada una vez por sesión. Para realizar lo que acabo de describir, y para implementar otras medidas de seguridad, publicaré otro post en un futuro muy cercano.

A pesar de los riesgos de seguridad existentes, estoy seguro que la gran mayoría de usuarios preferirán usar el método descrito en este post, y por lo tanto no cifrar la clave privada. Por esto he preferido escribir el post de esta forma y dejar el método técnicamente más seguro para más adelante.

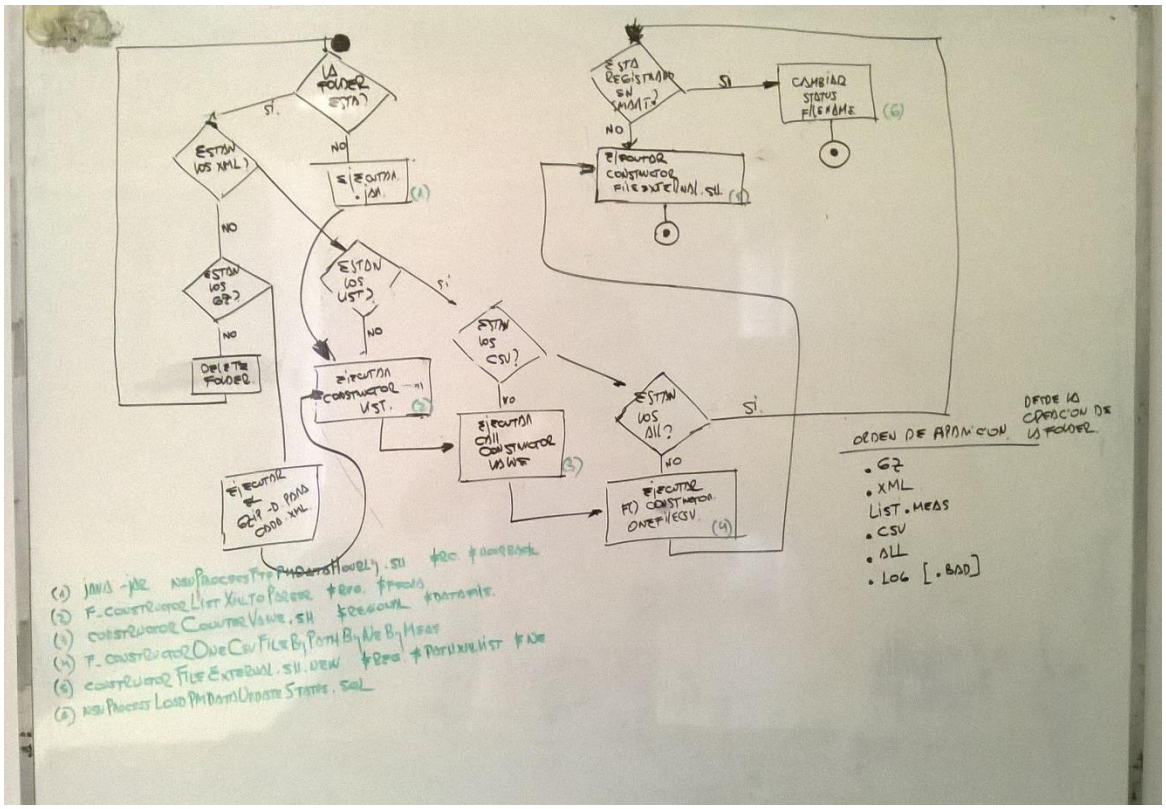
Estructura del proceso



Grafica A.2.1. Esquema general de la solución. Partiendo de la diana en color azul con el nombre de "Start Process" y concluyendo en la diana con color azul "End Process", en el medio, cada unos de los "steps" marcados en verde.

Cada paso del proceso esta subrayado en verde. En el recuadro de la izquierda están explicados los tipos de archivo de entrada y los tipos de archivos de salida, explicados paso por paso.
Es un esquema a mano hecho para explicar brevemente la arquitectura de la solución.

Guía Troubleshooting.



Grafica A.3.1. Diagrama de Flujo armado en unas de las capacitaciones que luego sirvió, para la construcción de la guía de Troubleshooting.

Este es el diagrama de flujo propuesto para describir las posibles situaciones y cuál es la solución al problema.

Por ultimo, el contenido de la guía de resolución propuesta…

```
TroubleShooting Parser

No estan los XML?
Verificar si la carpeta existe y si estan los GZ. Si los GZ estan, quiere decir que nunca se
descomprimieron.
Descomprimir y continuar con el procesamiento normal.

Como se descomprime?
Usando el comando "gzip -drf /calidad/data/nsn/storage/xml/rc1/pm/2016011208/"
mas info utiliza el comando "man gzip".

Precaucion: Tener cuidado con los permisos en el directorio, como en los files.
Es recomendable un "chmod -R 777" antes de cualquier accion, para evitar fallas.

Que se ejecuta una vez descomprimido?
Se ejecuta el callConstructorCounterValue.sh

Que hace el callConstructorCounterValue?
Todo. Partiendo de los XML, arma todos los csv, los all y terminando insertando en la statusProcessEtl.

Que pasa si tengo la mitad de los CSV generados.
Correria nuevamente el callConstructorCounterValue.
Los parametros de callConstructorCounterValue son el regional (1, 2, 3) y el folder (2016020415)

Que hacer si no estan todos los .all? Los 35?
Verificar primero la cantidad de xml, con "ls -l *.xml | wc".
cargar en memoria la funcion f_constructorOneCsvFileByPathByNeByMea2 y ejecutarla

la funcion f_constructorOneCsvFileByPathByNeByMea2 tiene dos parametros:
uno es el path completo (NO OLVIDAR EL ULTIMO / PORQUE NO ANDA SINO) y otro es el NE (WCEL, BTS,
LNCEL)
Esto es siempre y cuando esten los CSV.
Si los CSV no estan, igual que el punto anterior. Ejecutar callConstructorCounterValue.

Luego, queda ejecutar constructorFileExternal.sh que tiene tres parametros: regional (1,2,3),
PathCompleto (incluyendo el /) y el NE (WCEL, BTS; LNCEL).
Antes, si existe algun .all cargado en la tabla statusProcessEtl, borrarlo porque queda repetido.

Nota Extra:
En /process esta el reporte countHourMeas.sh que es muy util y tiene cuatro parametros: fecha,
regional, ne, meas.
```