UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Propuesta de una solución de monitoreo para sistemas del CeSPI
Autores: Otarán Federico y Perera Nicanor
Director: Luengo Miguel
Codirector: Rodriguez Christian Adrián
Carrera: Licenciatura en Sistemas y Licenciatura en Informática

Resumen

Los desarrolladores de sistemas informáticos de la Dirección de Desarrollo del Centro Superior para el Procesamiento de la Información (CeSPI), construyen aplicaciones para clientes internos y externos a la UNLP.Parte importante de la implementación y mantenimiento de los proyectos de la oficina es la recolección y análisis de datos de su funcionamiento. Este análisis es importante para conocer la salud de los servidores y el rendimiento de los equipos, detectar fallas en el hardware y software, descubrir comportamientos que afecten la funcionalidad de las aplicaciones y tomar decisiones que mejoren la productividad y calidad de los servicios brindados. En la actualidad existen una gran variedad de herramientas que pueden ser usadas para tener un seguimiento de métricas sobre servicios de software, y es posible integrar estas herramientas para crear un sistema que permita obtener, almacenar y visualizar los resultados.
El objetivo que tiene la investigación de esta tesis es plasmar el resultado de comparar y proponer una estrategia de recolección, análisis y utilización de información de la infraestructura y aplicaciones del CeSPI, que permita simplificar y enriquecer su análisis posterior y resulte en un monitoreo eficiente. *Se buscará alcanzar una solución de monitoreo que permita entender a la infraestructura no como piezas separadas, sino como un conjunto de componentes correlacionados.*

Palabras Claves

Monitoreo, estadìsticas, análisis de datos, series de tiempo, visualización de datos, alertas, DevOps

Trabajos Realizados

Investigación teórica de conceptos relacionados al monitoreo. Análisis del estado de arte de herramientas informáticas de monitoreo. Instalación y configuración de herramientas de recolección, almacenamiento, análisis y visualización de datos. Comparación de herramientas de alertas. Estudio y propuesta de metodologías.

Conclusiones

Se ha logrado construir una solución de monitoreo que se ajusta a la escalabilidad, dinamismo y automatización con los que cuenta la infraestructura del CeSPI y que puede utilizarse para armar tableros y alertas que permitan realizar un control básico y efectivo de las aplicaciones y mejorar el proceso de toma de decisiones. Hemos aprendido acerca de la importancia del monitoreo para resolver problemas rápidamente, estudiar el comportamiento de los sistemas y tomar decisiones para mejorar la implementación de los procesos de desarrollo, la infraestructura de software y las aplicaciones.

Trabajos Futuros

Monitoreo del propio sistema de alertas y del despliegue de aplicaciones. Estudio de herramientas alternativas. Sistemas de monitoreo que aprendan sobre si mismos. Monitorización de la infraestructura de monitoreo misma.

Fecha de la presentación: Abril 2017

Universidad Nacional de La Plata

Facultad de Informática

Tesina de la Licenciatura

Propuesta de una solución de Monitoreo para sistemas del CeSPI

Perera, Nicanor
Otarán, Federico

Director: Luengo, Miguel
Asesor Profesional: Rodriguez, Christian Adrián

17 de abril de 2017

Índice

Introducción

Los desarrolladores de sistemas informáticos de la Dirección de Desarrollo del Centro Superior para el Procesamiento de la Información (CeSPI), Universidad Nacional de La Plata (UNLP), construyen aplicaciones para clientes internos y externos a la UNLP. En esta oficina se aplican técnicas DevOps para agilizar el despliegue y mantener en sintonía los ambientes de desarrollo, testing y producción. Su equipo de trabajo está integrado por más de una docena de profesionales en informática, los cuales están a cargo de diferentes proyectos.

Parte importante de la implementación y mantenimiento de estos sistemas es la recolección y análisis de datos de su funcionamiento. Este análisis es importante para conocer la salud de los servidores y el rendimiento de los equipos, detectar fallas en el *hardware* y *software*, descubrir comportamientos que afecten la funcionalidad de las aplicaciones y tomar decisiones que mejoren la productividad y calidad de los servicios brindados.

Recoger datos y calcular estadísticas sobre servidores, aplicaciones y tráfico puede aumentar la seguridad en la toma de decisiones y permitir la anticipación a problemas.

La disciplina de recolectar datos de forma constante a lo largo del tiempo facilita la comprensión de los eventos que ocurren en cada instante, permitiendo contrastar un valor actual con valores previos. Esto proporciona la capacidad de respaldar decisiones con datos reales.

En la actualidad existen una gran variedad de herramientas que pueden ser usadas para tener un seguimiento de métricas sobre servicios de *software*, y es posible integrar estas herramientas para crear un sistema que permita obtener, almacenar y visualizar los resultados. Hoy en día, no todos los proyectos dentro del CeSPI cuentan con herramientas adecuadas para esta tarea.

Objetivo

El objetivo que tiene la investigación de esta tesis es plasmar el resultado de comparar y proponer una estrategia de recolección, análisis y utilización de información de la infraestructura y las aplicaciones del CeSPI, que permita simplificar y enriquecer su análisis posterior y resulte en un monitoreo eficiente.

Se espera que cualquier problema con las aplicaciones pueda ser detectado y analizado en un contexto global, y que diferentes fuentes de registros de datos puedan ser correlacionadas para aprender aún más sobre el estado de los proyectos. Para lograr esto realizará una evaluación el estado de arte de las estrategias de recolección de datos, estadísticas, monitoreo y generación de alertas. A partir de este análisis, habrá que relevar las herramientas que permitan implementar dichas estrategias, permitiéndo así proceder hacia un diseño de una infraestructura que las utilice.

Finalmente se integrará el sistema de monitoreo a la infraestructura actual de forma automatizada mediante prácticas DevOps, y buscará sentar las bases para que se puedan obtener tableros de control que permitan visualizar el estado de la infraestructura completa y que permita profundizar en resultados según los criterios de los diferentes perfiles de usuario.

Se busca entonces con este procedimiento identificar los datos de importancia a ser recolectados para alimentar un sistema de monitoreo que permita alertar de forma inteligente cuando alguna parte crítica del sistema se vuelva inestable o no responda a los resultados deseables. A su vez, se quiere conseguir la disponibilidad en todo momento de una vista de alto nivel del funcionamiento de toda la infraestructura, que facilite la toma de decisiones en distintos niveles según los intereses de los usuarios.

Teniendo en cuenta todo lo mencionado hasta ahora, se buscará alcanzar una solución de monitoreo que permita entender a la infraestructura no como piezas separadas, sino como un conjunto de componentes correlacionados.

Estructura del documento

En el capítulo 1 se muestran y comparan distintas definiciones de monitoreo, explicando dónde radica su importancia, qué información podría ser útil a los distintos perfiles de usuarios y cómo diseñar un sistema de monitoreo considerando esa información. Además, se explican algunos conceptos importantes que están íntegramente relacionados con este tema.

En el capítulo 2 se expone el caso de estudio en el que se centra la implementación sugerida. La metodología de trabajo del CeSPI, las tecnologías que forman parte de la infraestructura de las aplicaciones, el estado actual de la herramienta de monitoreo y los objetivos a la hora de implementar la solución son los temas principales en este capítulo.

En el capítulo 3 se explica cómo filtrar y almacenar datos obtenidos en tiempo real con el objetivo de poder utilizarlos más adelante. Se aborda la

recolección de datos de red, de recursos de los host, de los procesos y de las aplicaciones, entre otros. Además, se va a describir las distintas tecnologías y herramientas usadas.

En el capítulo 4 se plantea cómo implementar la obtención de datos de los *logs* de las aplicaciones y de los servicios. Describir los problemas comunes en la recolección de datos de *logs* y las herramientas utilizadas para resolverlos es el objetivo específico para este capítulo.

En el capítulo 5 se explica para qué sirve la visualización de datos, cuestiones importantes a tener en cuenta, la ventaja que da la visualización para el entendimiento de los datos, herramientas utilizadas y los resultados obtenidos.

En el capítulo 6 se describe qué son las alertas, las dificultades de implementar un buen sistema de alertas y las herramientas utilizadas junto con los resultados obtenidos.

Finalmente se cuentan las conclusiones de la experiencia que hemos obtenido a lo largo del desarrollo del trabajo y de nombran posibles trabajos futuros que se pueden realizar tomando como base esta investigación.

En el anexo A se desarrollan aspectos de algunas herramientas que utilizan en el CeSPI.

En el anexo B se mencionan otras herramientas que si bien no forman parte de nuestra solución final, han formado parte de la investigación y que pueden ser útiles para soluciones similares.

En el anexo C se amplía el código utilizado en una parte de nuestra solución.

1. Marco Teórico

No existe, en el ambiente de la informática, un acuerdo común sobre la definición precisa de monitoreo. Es por esta razón que en la sección 1.1 se analizarán las opiniones de varios autores sobre el significado de este término. A partir de estas definiciones se intentará evidenciar la importancia de la aplicación de monitoreo en proyectos de *software*.

Para poder comprender algunos de los pasajes de los capítulos siguientes y brindar la base teórica de este trabajo, en la sección 1.2 se nombrarán las definiciones de algunos términos importantes, incluyendo métricas, series de tiempo, estadísticos, intervalos de tiempo, granularidad, percentiles y distribución de frecuencias, entre otros.

El proceso de monitoreo comienza con la acumulación de datos por agentes de recolección. En la sección 1.3 se dará la definición de agente de recolección, y se enumeraran los distintos tipos de agentes que existen. Además se indagará en cómo las necesidades de diferentes tipos de usuario determinan los datos a recolectar.

Se denomina *log* al registro de una operación en una computadora, usualmente almacenado en un archivo. En la sección 1.4 se profundizará en la definición de *log* y se abordará su utilidad para el monitoreo. Además se explicarán las partes de un *log*, su formato y las dificultades más comunes a la hora de manipular este tipo de registros.

La visualización de datos es el proceso de interpretación, contrastación y comparación de datos que ofrece un conocimiento en profundidad y detalle de los mismos de tal forma que se transformen en información comprensible para las personas. En la sección 1.5 se ahondará más en este tema.

Finalmente, en la sección 1.6 se definirá el concepto de alertas en el contexto de monitoreo, se pondrá en manifiesto la importancia de las mismas y se describirán los obstáculos más comunes a la hora de implementar una solución de este tipo.

1.1. Monitoreo

En su libro Effective monitoring and Alerting for Web Operations, Stawek Ligus define al monitoreo como el proceso de mantener en constante observación la existencia y magnitud de los cambios de estado y el flujo de datos de un sistema, y que tiene como objetivo identificar los fallos y ayudar

en su posterior eliminación[9, p. 2].

Según el autor, las técnicas utilizadas en el monitoreo de los sistemas, son compartidas con aquellas técnicas usadas en los campos de procesamiento de datos en tiempo real, estadísticas y análisis de datos.

El escritor a su vez define sistema de monitoreo como el conjunto de componentes de *software* utilizados para la recolección de datos, su procesamiento y su presentación.

En el libro Art of monitoring, James Turnbull explica de forma sencilla el proceso por el cual el monitoreo traduce las métricas tomadas de diferentes fuentes en experiencia de usuario medible. Esta experiencia de usuario proporciona retroalimentación a la empresa u organización para ayudar a que la entrega se transforme verdaderamente en lo que los clientes quieren, y también a los desarrolladores y encargados del monitoreo indicando si algún componente del sistema no funciona o si la calidad del producto final no es aceptable [20, p. 8].

El autor resalta la importancia de la recolección, procesamiento y análisis de los datos en un sistema y explica de forma simplificada lo mencionado anteriormente al manifestar que un sistema de monitoreo se encarga de traducir las métricas generadas por los sistemas y aplicaciones en información de valor para el negocio.

Jason Dixon define, en su libro Monitoring with Graphite, al monitoreo como el conjunto de *software* y procesos que son utilizados para asegurar la disponibilidad y salud de uno o más sistemas y servicios, y en un sentido abstracto afirma que puede ser dividido en tres grandes categorías: detección de fallas, producción de alertas y planificación de crecimiento.

La detección de fallas consiste en identificar cuando un recurso o colección de recursos deja de estar disponible o disminuye su rendimiento. Se suelen usar umbrales para identificar variaciones importantes en el comportamiento del sistema. Hoy en día además se utilizan técnicas de aprendizaje automática para la detección de anomalías.

La producción de alertas es importante para dar a conocer sucesos importantes o que requieran atención inmediata. Cuando ocurre un problema es deseable que se notifique a través de un mensaje al responsable de solucionarlo.

La planificación de crecimiento consiste en estar preparado para cambios futuros. Esto se puede lograr a partir del seguimiento de las métricas del servidor y las aplicaciones. Es acerca de la habilidad de estudiar las tendencias

de los datos y usar ese conocimiento para tomar decisiones informadas [3, p. 15].

Se tomarán estas miradas como punto de partida para hacer un análisis más profundo acerca de los aspectos más importantes a tener en cuenta a la hora de diseñar un sistema de monitoreo eficaz.

1.2. Métricas y series de tiempo

Las métricas de monitoreo son colecciones de entradas de datos numéricos organizadas en grupos de listas consecutivas y cronológicamente ordenadas. Cada entrada de datos consiste en un valor de medición registrado, la marca de tiempo en la que tuvo lugar la medición y un conjunto de propiedades que la describen.

Cuando las entradas de datos de una métrica son segmentadas en intervalos de tiempo fijos y convertidas en un estadístico mediante una transformación matemática, pueden ser presentadas como series de tiempo e interpretadas en gráficos bidimensionales.

El largo de los intervalos entre valores, también llamado granularidad, depende de los tipos de mediciones y el tipo de información que se quiere extraer. Intervalos comunes incluyen 1, 5, 15 y 60 minutos, pero también es posible mostrar intervalos tan pequeños como un segundo y tan extensos como un día.

La ventaja más importante de usar datos de serie de tiempo para el monitoreo es su propiedad de ilustrar con precisión el proceso de cambio en un contexto de datos históricos. Son una herramienta indispensable para encontrar la respuesta a la pregunta crítica: ¿Qué ha cambiado y cuándo?

Los datos de este tipo tienen dos dimensiones: una de datos propiamente dicha y otra de tiempo. Esto significa que dos registros independientes compartirán siempre la dimensión de tiempo. Es por esto que este tipo de registros son útiles para destacar relaciones correlativas entre datos de muchas fuentes.

Estadísticos

Las métricas de monitoreo almacenan las entradas de datos y describen sus propiedades. El proceso de generar y graficar series de tiempo implica recuperar un subconjunto de entradas de datos especificando un conjunto de

propiedades, dividiéndolas en intervalos de tiempo espaciados uniformemente, y matemáticamente resumir entradas de datos en cada intervalo. Esto es realizado con el uso de estadísticos. Algunos de los estadísticos de uso más común son:

- La cantidad de entradas por intervalo.
- La suma de valores de todas las entradas.
- El valor promedio de todas las entradas.
- Los percentiles[1] de los valores entrada, incluyendo el mínimo[2], el máximo[3] y la mediana[4].
- La desviación estándar del promedio en la distribución de entradas recolectadas.

Los estadísticos describen conjuntos de entradas observadas por sus centros (promedio o mediana), el total (suma o cantidad), y la distribución y propagación (percentiles y desviaciones). Pueden resumir grandes conjuntos de datos de forma compacta y concisa. Convertir muchos números en uno sólo causa la pérdida de información, pero el resumen suele ser lo suficientemente preciso como para sacar conclusiones confiables.

Distribución de frecuencia y percentiles

La distribución de frecuencia es un resumen de un conjunto de datos que combina elementos numéricos en grupos a través de un proceso conocido como *binning*. Esta distribución puede ser ilustrada como un histograma, que los presenta de manera que permite a las personas ver su tamaño relativo de forma rápida.

Los histogramas a menudo caen a lo largo de una distribución normal, con sus columnas encajando justo debajo de la famosa curva gaussiana. Pero los datos del sistema no suelen ser tan regulares y suelen mostrar una cola larga

[1] pueden tener valores de 0 a 100

[2] p0

[3] p100

[4] p50

del lado derecho; lo que significa que la distribución se encuentra inclinada hacia el lado izquierdo.

Hay una razón simple para eso: el límite inferior en el rendimiento es un límite duro: el mejor caso de tiempo de respuesta debe ser siempre más de cero unidades de tiempo. Su límite superior es uno blando: teóricamente se puede esperar por siempre.

Los estadísticos graficados como valores en el tiempo son convenientes para observar cambio, pero las series de tiempo no revelan necesariamente la verdadera naturaleza de los datos. Es posible extraer datos de entradas provenientes de análisis de logs sin conexión y presentarlos en un histograma para mostrar su frecuencia relativa. Esta información da a los operadores una buena idea de qué valor esperar de una entrada típica y cómo se ve la distribución de entradas detrás de cada valor resumido.

Una forma alternativa de resumir la distribución de frecuencias es un gráfico de percentiles. Para cualquier conjunto de entradas, un percentil es un número real en el rango de 0 a 100 con un valor correspondiente de ese conjunto. El número en un percentil particular muestra cuántos valores son más pequeños que el valor de ese percentil.

Los percentiles se calculan clasificando el conjunto de entradas por valor en orden ascendente, encontrando el rango para un percentil dado (es decir, la dirección del valor en la lista ordenada) y buscando el valor por rango.

El percentil 0 es la medida del valor más bajo (el primer elemento de una lista ordenada, o sea el mínimo), y el percentil 100 es el valor máximo registrado (el último elemento de la lista, o sea el máximo). Al percentil 50 se lo conoce comúnmente como la mediana, y representa el valor medio en el conjunto

Los percentiles facilitan la interpretación de la distribución; por ejemplo, para medir el tiempo de respuesta el valor de percentil 98 en 3 segundos significa que el 98 % de todas las solicitudes se completó en 3 segundos o menos. Por el contrario, el 2 % de las solicitudes más lentas tomaron 3 segundos o más.

Tasa de cambio

La tasa de cambio ilustra el grado de cambio entre valores en una serie de tiempo u otra curva. Efectivamente, cuando la pendiente del gráfico original está aumentando, su tasa de cambio tiene valores positivos. Por el contrario, cuando la pendiente desciende, los valores de la serie de tasas de cambios son

negativos.

La tasa de cambio derivada de una serie de tiempos puede ser presentada como otra serie de tiempo. La tasa de cambio es una herramienta conceptual útil para ilustrar niveles de crecimiento o disminución con el tiempo. Es usada extensivamente con métricas de conteo para expresar el número de incrementos del contador por intervalo de tiempo.

Granularidad de tiempo

Los datos en una función de serie de tiempos son presentados en una granularidad de tiempo fijo. Una granularidad fina se traduce en períodos cortos de valores. Cuanto más gruesa sea la granularidad, más largo será el período.

Las métricas de granularidad fina tienden a revelar el momento exacto en que ocurrió un evento y, por lo tanto, son útiles para describir líneas de tiempo y encontrar dirección en las relaciones causales. Sin embargo podrían ser más costosas de almacenar.

Las métricas de granularidad gruesa, por el otro lado, son mucho más adecuadas para ilustrar tendencias.

La selección de la granularidad correcta para presentar una métrica es importante para una interpretación precisa de los datos. Tanto las medidas demasiado granulares como las demasiado gruesas pueden oscurecer el punto que está tratando de transmitir.

Algunos sistemas de monitoreo usan intervalos constantes predefinidos y no permiten al usuario modificar su granularidad, mientras que otros sistemas permiten especificar períodos arbitrarios.

Sin embargo, siempre se requiere un intervalo mínimo. Incluso si uno fuera capaz de presentar eventos en una escala de tiempo continuo (con una granularidad infinitesimalmente fina), probablemente no sería muy útil.

En el mundo real no pueden ocurrir dos eventos exactamente al mismo tiempo, por lo que el registro de eventos en una escala continua no podría hacer que el número de eventos se acumulara en el gráfico.

Pasando al otro extremo, si el intervalo de datos es extremadamente largo, como un año, la salida será un pequeño número de enormes colecciones de eventos. Eso puede ser algo útil para los propósitos de análisis de datos pero no para el monitoreo.

Agregación de métricas

En sistemas distribuidos, las entradas de datos para la misma métrica provienen de muchas fuentes. Por ejemplo, en un grupo de servidores *web* detrás de un balanceador de carga, reportando estadísticas acerca de solicitudes siendo servidas, los datos de solicitud podrían ser vistos en la forma de múltiples funciones de series de tiempo, una por cada servidor *web*, graficadas unas contra otras. Sin embargo, la agregación podría combinar los resultados de los servidores *web* en una única secuencia de registros de serie de tiempos con un total de todas las solicitudes.

La agregación permite obtener una vista general de los datos y simplifica el gráfico, pero la capacidad de explorar para ver cada fuente de datos no es menos importante. Si uno de los servidores deja de tomar solicitudes, reportará valores en cero, o directamente dejará de enviar entradas por completo. La falla podría ser detectada mirando las métricas individuales del servidor, pero no necesariamente se mostrarían en el gráfico agregado.

Muchas fallas, sin embargo, son mucho más sutiles y se manifiestan a través de pequeñas depresiones en el número de peticiones en lugar de su desaparición completa. En estos casos, es también deseable agregar valores de todas las fuentes y presentarlos como una única métrica para mostrar el efecto acumulativo [9, p. 21-25].

1.3. Recolección de datos

El proceso de monitoreo comienza con la acumulación de datos por agentes de recolección.

Un agente de recolección es un programa de *software* especializado que capta información significativa sobre entidades monitoreadas como pueden ser distintos tipos de terminales, bases de datos o dispositivos de red, etc., la encapsula en entradas de datos cuantitativos y reporta estas entradas a un sistema de monitoreo en intervalos regulares.

Estas entradas son transformadas en métricas que pueden ser almacenadas en series temporales, es decir, secuencias de datos, observaciones o valores ordenados cronológicamente.

La recolección de datos puede ser un proceso continuo o también puede darse de forma periódica en intervalos regulares de tiempo, dependiendo la naturaleza de las mediciones y del costo de los recursos involucrados en dicha recolección de datos.

Según el conocimiento de la implementación de las entidades a monitorear, los agentes de recolección pueden clasificarse en caja blanca o caja negra.

Los agentes de recolección de caja blanca son aquellos que trabajan sobre la estructura interna de las entidades a monitorear. Entre ellos podemos encontrar:

- **Log Parsers:** los analizadores gramaticales o parsers de *logs* extraen información específica proveniente de las entradas de los *logs*, tales como códigos de estado y tiempos de respuesta de las peticiones desde los *logs* de los servidores *web*.
- **Log Scanners:** los escaneadores de *logs* cuentan las ocurrencias de una cadena de caracteres en las líneas de los archivos de *logs* definidos por una expresión regular. Por ejemplo, para obtener información de errores regulares y críticos es posible calcular el número de ocurrencias de la expresión regular `/ERROR|CRITICAL/` dentro de un archivo de *logs*.
- **Interface readers:** los lectores de interfaces leen e interpretan interfaces de sistemas y dispositivos. Un ejemplo son los lectores de temperatura y humedad de dispositivos especializados, o el pseudo sistema de ficheros `/proc`, que permite acceso a información del kernel sobre los procesos en sistemas UNIX.

Los agentes de recolección de caja negra son aquellos que toman información sin adentrarse en la implementación de dichas entidades. Entre ellos se encuentran:

- **Probers:** son programas que se ejecutan fuera del sistema a monitorear enviando peticiones al sistema en cuestión para confirmar su respuesta. De esta forma trabajan las peticiones ping. Otro ejemplo de este tipo de agentes son las solicitudes *Hypertext Transfer Protocol* (HTTP) hacia un sitio *web* con el propósito de verificar su disponibilidad.
- **Sniffers:** son programas informáticos que registran la información que envían los periféricos. Por ejemplo, pueden monitorear interfaces de red y analizar estadísticas de tráfico como el número de paquetes transmitidos bajo un protocolo. [9, p. 15-16]

En el marco de una infraestructura de *software*, los datos a recolectar por un agente pueden ser obtenidos de varias fuentes. Por ejemplo, de una

computadora es posible obtener información del uso de *Central Processing Unit* (CPU), espacio de disco y memoria. A partir del tráfico *web* es posible medir el uso del ancho de banda por usuario, aplicación, protocolo y grupo de direcciones *Internet Protocol* (IP). Se puede medir el rendimiento y uso de una aplicación *web*, y también reglas de negocio, como por ejemplo cantidad de hitos alcanzados.

Estas métricas pueden ser útiles para usuarios con diferentes necesidades. Por ejemplo, un sistema de monitoreo podría ser usado por un gerente, un líder de proyecto, un desarrollador, un profesional de *Information Technology* (IT), un analista de datos y un cliente.

Teniendo en cuenta el rol que cumple una persona en un proyecto, es posible averiguar qué información puede serle útil y a partir de allí elegir el mejor método para almacenar esta información, analizar cómo construirla a partir de los datos, localizarlos y estudiar cómo obtenerlos.

Al profesional de IT le podría servir tener un tablero donde se observe la salud de los sistemas en tiempo real. Además podría interesarle conocer la disponibilidad de los servicios, la cantidad de tráfico en la red, el uso de CPU y memoria, el uso de memoria swap y la cantidad de espacio de disco utilizado.

Un líder de proyecto puede querer disponer de gráficos donde se visualice la cantidad de despliegues por semana, el número de problemas resueltos por los programadores, el rendimiento de la aplicación y el tiempo promedio de respuesta de la aplicación.

Un desarrollador puede encontrar utilidad en la visibilización del rendimiento de diferentes funcionalidades de la aplicación, la eficiencia en tiempo de los accesos a las bases de datos y la cantidad de accesos a la aplicación en distintos períodos de tiempo. Además puede interesarle contar con información del comportamiento de los recursos del servidor en el ambiente productivo.

Se puede decir que al desarrollador le interesan principalmente las métricas de aplicación. En cambio, al propietario de un sistema pueden interesarle más las métricas de negocio.[20, p. 446]

Las métricas de negocio están muy relacionadas con las métricas de la aplicación. Por ejemplo, si se piensa medir el tiempo de ejecución de una transacción de pago como métrica de aplicación, podrían obtenerse las siguientes métricas de negocio con el contenido de estas solicitudes:

- número de nuevos usuarios o clientes

- número de ventas
- ventas por valor o ubicación

Hay que tener en cuenta, como dice autor en The Art Of Monitoring, que "El primer cliente de su sistema de monitoreo es el negocio. El monitoreo existe para apoyar el negocio y para asegurarse de que continúa haciendo su trabajo." [20, p. 8].

1.4. Utilidad de los logs

Se denomina *log* al registro de una operación en una computadora. Los *logs* son normalmente guardados en un archivo. Los *logs* se utilizan para describir un conjunto de acontecimientos, tales como: accesos de usuarios a un sistema, manipulación de datos, auditoría, diagnóstico de dispositivos y medidas de seguridad.

Un registro de *log* contiene normalmente una marca temporal, la fuente de la información y el mensaje o contenido. La marca temporal, también conocida como *timestamp*, es una secuencia de caracteres que indica la hora y fecha en la que ocurrió un evento. La fuente es la aplicación, programa o sistema que generó el mensaje. Muchas veces la fuente es representada por la IP o el nombre de un host y el nombre de la aplicación.

Los *logs* pueden ser de gran utilidad para visualizar información de recursos del sistema, de usuarios de las aplicaciones y del funcionamiento de las mismas aplicaciones. Sin embargo, los *logs* como fuente de información suelen ser subestimados por desarrolladores y los profecionesles de IT.

A menudo los *logs* son completamente ignorados y solamente tomados en cuenta en situaciones de problemas en las aplicaciones o el sistema, y normalmente se los elimina sin haber sido revisados antes [1, p. 16].

Los *logs* pueden ser catalogados según su utilidad de varias maneras. Cada aplicación define como clasificar los *logs* según su relevancia. Por ejemplo, Syslog, el estándar de más popular para el envío de mensajes de registro en redes, define los siguientes niveles de gravedad para sus *logs*[5]:

Si bien no todas las herramientas que implementan *logs* utilizan todos lo niveles de severidad anteriormente mencionados, la mayor parte de ellos

[5] `https://tools.ietf.org/html/rfc3164`

Cód.	Gravedad	Descripción
0	Emergency	El sistema se encuentra inutilizable
1	Alert	Una acción debe ser tomada inmediatamente
2	Critical	Condición crítica
3	Error	Condición de error
4	Warning	Condición de advertencia
5	Notice	Condición normal pero significativa
6	Informational	Mensaje informativo
7	Debug	Mensajes de bajo nivel

Cuadro 1: Tabla de tipos de logs para Syslog.

toman esta tabla como referencia.

A continuación se explicará algunos de estos niveles de relevancia:

- Los *logs* de información proveen evidencia documentada de una secuencia de actividades que afectan en algún momento a una operación específica, procedimiento o evento. Permiten a los desarrolladores y administradores advertir la ocurrencia de un evento benigno, como por ejemplo el inicio de sesión de un usuario en un sistema.
- Los registros de depuración son generados por los desarrolladores con la finalidad de entender la actividad de un sistema, identificar problemas en el funcionamiento del código de la aplicación y encontrar soluciones para dichos problemas.
- Los mensajes de advertencia indican comportamientos inesperados o situaciones que deben ser vigiladas por un programador pero que no tienen un impacto negativo en el funcionamiento normal del sistema. Un ejemplo es mensaje aconsejando la descontinuación del uso de una funcionalidad que será eliminada en futuras versiones de un *software*.
- La etiqueta de error se usa para transmitir fallos que se producen en un sistema informático, y que requieren la atención inmediata de un programador o persona especializada. Indican que existe la posibilidad de que parte de la aplicación no esté funcionando[1, p. 3].

Es conveniente que los desarrolladores hagan buen uso de los niveles de severidad de los *logs* para beneficiarse de la información que estos puedan brindar acerca de las aplicaciones o sistemas. Si las etiquetas de relevancia son utilizadas correctamente, se logran datos de mejor calidad que facilitan el análisis posterior de los registros.

Retos del manejo de logs

Existen varios desafíos que la mayoría de las organizaciones deben enfrentar a la hora de gestionar los *logs*.

Uno de estos desafíos es la existencia de numerosas fuentes de *software* generadores de registros de *log*. Muchas veces estas fuentes se encuentran ubicadas en diferentes hosts, y una fuente puede generar diferentes tipos de registros, por ejemplo almacenar los intentos de autenticación en una aplicación en un archivo de *logs* y la información relacionada con la ocupación de la red en otro archivo distinto.

Otro reto que presenta el manejo de *logs* es el hecho de que los mismos registran a menudo sólo las piezas de información que se consideran más importantes para un evento particular. Esto puede dificultar analizar la correlación de eventos registrados por diferentes fuentes de *logs*, ya que pueden no tener valores comunes registrados.

Estas diferencias pueden ser leves, como la utilización de formato de fechas `MMDDYYYY` en un archivo de *logs* y el uso de `MM-DD-YYYY` en otro. Pero también pueden ser complejas, como la identificación de un servicio por el nombre en un *log*, y la identificación de un servicio por el número de puerto en otro.

Otro obstáculo en la gestión de *logs* es la inconsistencia de las marcas temporales. Cada host que genera *logs* suele hacer referencia a su reloj interno. Si el reloj de un host es inexacto, el *timestamp* que genera en los *logs* también lo es.

Esto puede dificultar el análisis de los *logs*, particularmente cuando se analizan *logs* de varias computadoras. Por ejemplo podría suceder que la marca temporal de un evento A indique que dicho evento haya ocurrido 45 segundos antes que un evento B, cuando en realidad el evento B haya ocurrido varios minutos antes que el evento A.

Otra complicación en la administración de *logs* es que distintas fuentes utilicen diferentes formatos para almacenar sus *logs*. Por ejemplo, algunas herramientas pueden guardar sus *logs* como archivos de texto separados por comas o por *tabs*, utilizando el formato Syslog o a través de *Simple Network Management Protocol* (SNMP). También pueden existir sistemas que almacenen *logs* con formato *Extensible Markup Language* (XML), *JavaScript Object Notation* (JSON) o como archivos binarios.

Algunos registros de *logs* han sido diseñados para ser fácilmente leídos por seres humanos, y otros para ser fácilmente analizados por computado-

ras. Existen programas que usan formatos estándares para almacenar *logs*, y también existen otros que usan configuraciones propietarias. Además hay *logs* que no han sido diseñados para su almacenamiento local en un archivo, sino para ser transmitidos a un sistema de procesamiento, como por ejemplo los *routers* y *switches* que implementan el protocolo SNMP[8, p. 23] anteriormente mensionado.

La naturaleza distribuida de los *logs*, el uso de formatos inconsistentes y el considerable volumen de información almacenado en los mismos consiguen que la gestión de la creación y almacenamiento sea un desafío.

Los administradores de las redes y sistemas son en la mayoría de las organizaciones los responsables de realizar el análisis de *logs*.

Este análisis es tratado a menudo como una tarea de baja prioridad por parte de los administradores, y la razón de esto es que otras funciones como la solución de problemas operativos y la atención a vulnerabilidades de seguridad suelen requerir respuestas rápidas. Además, los administradores generalmente no cuentan con herramientas eficaces que permitan automatizar el proceso de análisis.

Los *logs* pueden ser realmente muy útiles si se los gestiona de manera correcta. Por ejemplo, pueden describir excepcionalidades que ocurran en nuestras infraestructuras, permitir el análisis del rendimiento de las aplicaciones y detectar errores en el sistema. También pueden ser una buena fuente de información para determinar las causas de un incidente.

Además de describir un problema, los *logs* pueden ser beneficiosos para descubrir si se han tomado buenas decisiones de diseño de las aplicaciones y prevenir problemas futuros. Asimismo, un *log* puede contener información importante para generar alertas. En concreto, si los *logs* de un servidor *web* retorna un error HTTP con código 500, es porque una aplicación no está funcionando correctamente [1, p. 30-31].

1.5. Visualización efectiva

La visualización de datos es el conjunto de técnicas usadas para comunicar datos o información a través de su codificación como objetos visuales contenidos en gráficos. La meta es comunicar información de forma clara y eficiente a los usuarios. La visualización efectiva ayuda a los usuarios a analizar y razonar sobre datos y evidencia, y transforma datos complejos en datos accesibles, entendibles y utilizables.

Complaint Rate	Mean	Median	Top Quartile	Bottom Quartile
Overall	0.037%	0.001%	0.00100%	0.995%
U. S.	0.028%	0.003%	0.00010%	0.897%
EMEA	0.030%	0.000%	0.00010%	0.152%
United Kingdom	0.023%	0.100%	0.00100%	0.279%
Canada	0.021%	0.007%	0.00600%	0.085%
APAC	0.014%	0.040%	0.00000%	0.126%

Figura 1: Diagrama de representación en forma de tabla

El uso de técnicas efectivas de visualización permite entender mejor los datos y ayuda a descubrir tendencias, revelar ideas, explorar fuentes y contar historias. En los últimos años la visualización de datos se ha vuelto un área activa de investigación, enseñanza y desarrollo.

Para este desarrollo nos interesa usar técnicas de visualización de datos para elaborar cuadros de mando útiles y poder comunicar información de la manera más adecuada para cada usuario.

La visualización de datos es una herramienta para ayudar al análisis y no un sustituto de la habilidad analítica. Para lograr una visualización de datos efectiva es útil contar con conocimiento de negocio, estadística, teoría del color, psicología, composición gráfica e inteligencia emocional.

Existen muchas formas de presentación de datos, y cada una tiene su propósito específico. A continuación se explican algunas de ellas:

Tabla

Es un modo de organizar información en filas y columnas que es útil para comparar valores. A través del uso de colores y explicaciones, se puede hacer que la información contenida en las mismas sea más fácil de decodificar (Figura 1).

Gráfico de barra

Es una forma de representar gráficamente conjuntos de datos o valores, a través del uso de barras rectangulares de longitudes proporcionales a los valores representados. Son útiles para comparar elementos (Figura 2).

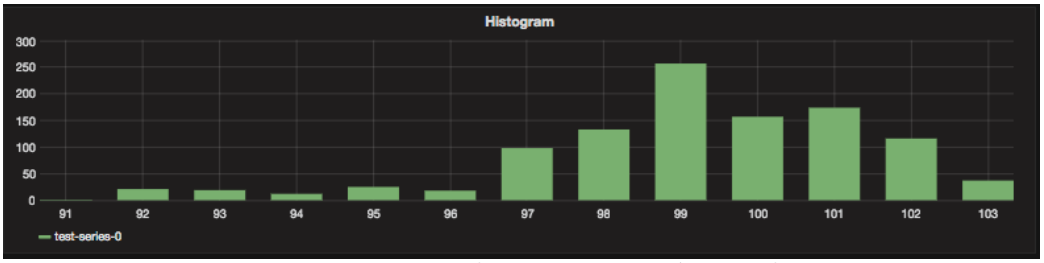

Figura 2: Diagrama de representación con barras

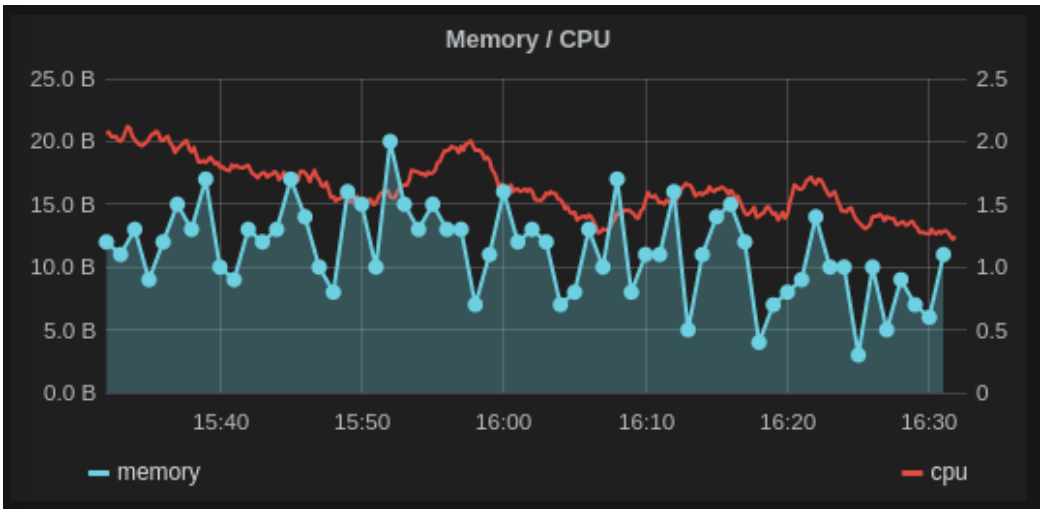

Figura 3: Diagrama de representación con líneas

Gráfico de línea

Es una manera de visualizar información compuesta de una serie de datos representados por puntos y unidos por segmentos lineales. Este tipo de gráficos sirve para comprobar de forma sencilla la tendencia de los datos.(Figura 3)

Gráfico circular

También llamado gráfico de torta o *pie chart*. Es un recurso estadístico que se utiliza para representar porcentajes o proporciones. Consiste en dividir a un círculo en secciones coloreadas, donde el tamaño de cada sección es proporcional al valor que se intenta representar (Figura 4).

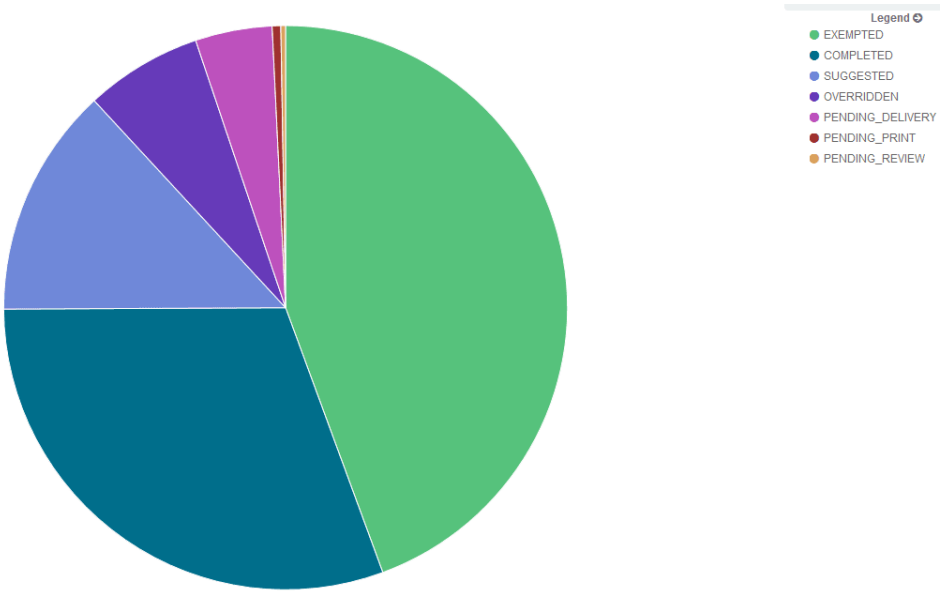

Figura 4: Diagrama de representación en forma circular

Figura 5: Diagrama de representación de dispersión

Gráfico de dispersión

También conocidos como *Scatter Plot*, estos gráficos usan valores numéricos para ambos ejes. Son útiles para mostrar la relación entre distintos tipos de datos. Los gráficos de burbujas son una variación de este tipo de gráficos, los puntos pueden ser de distintos tamaños, representando una dimensión adicional a los datos (Figura 5).

A la hora de mostrar información es importante conocer las distintas maneras de visualizar información, analizar qué es lo que se quiere transmitir y tener criterio para decidir la forma en la que se mostrarán los datos.

1.6. Alertas

En las secciones anteriores se analizaron conceptos importantes en cuanto a la obtención, manipulación, procesamiento, almacenamiento, resumen y visualización de información de sistemas que proveen información valiosa.

En el libro Effective Monitoring and Alerting, el autor explica que con la información generada a través de técnicas de monitorización es posible producir un gran número de tableros a los cuales es importante prestarles atención. Contratar personas para visualizar estos tableros no siempre es una tarea rentable. Esta tarea no es muy gratificante, e incluso si lo fuera, es por lo menos discutible que un operador humano sea mejor que una máquina o algoritmo reconociendo patrones cuya atención pueda prevenir una situación no deseada [9, p. 47].

Una alerta puede ser un mensaje enviado con el fin de notificar sobre un evento importante a una persona. Este mensaje puede ser transmitido mediante correo electrónico, servicio de mensajes simples, mensaje instantáneo, llamada telefónica o a través de un servicio de notificaciones de alguna herramienta de *software*.

Es deseable que un sistema de monitoreo tenga la capacidad de detectar aquellos eventos significativos o que denoten un grave cambio de estado en el negocio, aplicación o sistema, y puedo notificarlo a los interesados a través de un sistema de alertas.

Los sistemas de alertas suelen ser configurados por los profecionales de IT para detectar y prevenir problemas. Las alertas pueden brindar información sobre eventos no deseados, umbrales sobrepasados, caídas del sistema o hitos alcanzados.

Las alertas deben ser transmitidas al destinatario adecuado, es decir, a una persona que esté obligada a tratar con el evento.

Según Jason Dixon, autor de Monitoring with Graphite, para todos los propósitos prácticos, alertar constituye el momento en que un sistema de monitoreo identifica un fallo que requiere alguna acción adicional. En la mayoría de los casos, el sistema está diseñado para enviar un correo electrónico a un miembro del equipo de operaciones, pero conseguir que la alerta llegue a su destino no suele ser tan simple como parece.

Por ejemplo, puede suceder que el destinatario esté fuera del rango de su servicio celular, o que tenga la computadora apagada. Es por esto que la programación de llamadas y el enrutamiento de notificaciones deben ser incluidos en cualquier discusión de alertas.

A medida que una organización crece, es mayor el número de personas que pueden resolver el problema notificado, pero, al mismo tiempo, la complejidad para saber cómo planificar estas alertas aumenta.

En este punto entran en juego las herramientas de manejo de alertas. Las mismas se encargan de administrar alertas y gestionar los avisos. Algunas incluso permiten organizar un calendario de llamadas [3, p. 17].

Las alertas son registradas a menudo en la forma de una petición o *ticket* en un sistema de sistema de seguimiento de incidentes (*Issue Tracking System*).

Un sistema de seguimiento de incidentes es un paquete de *software* que administra y mantiene listas de incidentes conforme son requeridos por una institución. Estos sistemas suelen ser usados por personal de servicio al cliente para crear, actualizar y resolver incidentes reportados por usuarios, empleados de la organización o sistemas de alertas automatizados.

Stawek Ligus aconseja operar de acuerdo al principio de monitoreo extensivo y de alerta selectiva. Esto significa identificar qué métricas son útiles para el negocio y configurar alarmas para las series temporales a partir de indicadores clave. El autor sugiere una lista de métricas a considerar como candidatas para alertar:

- Recursos:

  Retardo de red y pérdida de paquetes: para obtener la métrica de latencia se puede utilizar el comando `ping` tanto a una dirección externa como a una dirección interna y registrar el tiempo de ida y vuelta de cada una. Además se puede utilizar la información de `ping` para calcular la pérdida de paquetes. Si el paquete llega con éxito se anota 0, y si no llega se anota 1. Calculando el promedio de estos valores se obtiene la pérdida de paquetes en términos porcentuales.

  Utilización de CPU: este valor sirve como indicador para el esfuerzo computacional de las operaciones. En un sistema Linux puede obtenerse analizando el directorio `/proc/stat`.

  Espacio disponible de disco y memoria: es importante evitar que el almacenamiento local se llene y que la cantidad de espacio libre en disco se aproxime a los límites. El uso excesivo del almacenamiento local puede inducir a un comportamiento impredecible para las aplicaciones.

- Plataforma:

Tiempo de respuesta: puede ser útil extraer y registrar las estadísticas de tiempo de respuesta de los logs de las aplicaciones. Particularmente puede ser importante prestarle atención a los cambios en el promedio y a los percentiles.

Códigos de respuesta: registrar los códigos de errores HTTP en aplicaciones *web* es una buena idea si se quiere alertar sobre una proporción inusual de códigos HTTP referidos a requerimientos erróneos o problemas del servidor.

- Aplicación:

  Disponibilidad: si se quiere lanzar una alerta cuando el porcentaje de solicitudes exitosas se encuentre debajo de un determinado umbral, es buena idea configurar controles de disponibilidad desde distintas ubicaciones, emitir solicitudes de prueba a cada minuto y registrar casos de éxito y de falla.

  Tasa de error: si se define de forma correcta aquello que constituye un error en las aplicaciones y se lo monitoriza muy de cerca, se puede decidir lanzar una alerta cuando estos errores suceden en una proporción relativamente alta.

Falsas alarmas

Un falso positivo es una alerta desencadenada por accidente o una mala configuración. Estas alertas pueden ser molestas para los desarrolladores, y además pueden conducirlos a desconfiar del sistema de monitoreo.

Si un programador tiene muchos falsos positivos, puede ocurrir que reste importancia a las alertas y pierda información importante. Por ejemplo, un sistema de alertas que envía gran cantidad de correos electrónicos con falsos positivos puede hacer que el usuario desestime las alertas, deje de prestarles atención y en consecuencia no actúe cuando ocurra un evento significativo que necesite su atención inmediata.

Un falso negativo es una alerta que nuestro sistema de monitoreo no puede detectar. Puede ser causada por el uso de umbrales inadecuados, por falta de controles o por la utilización de intervalos de verificación de mucha o de muy poca duración.

Los falsos negativos suelen ser identificados demasiado tarde, resultando por ejemplo en la caída de un sistema o en la discontinuidad de la disponibilidad de un servicio [3, p. 16].

Es muy importante, a la hora de armar una solución de alertas, tener muy presenta estas situaciones y reducir al máximo el número tanto de falsos positivos como negativos. Esto puede ser determinante para conseguir el éxito o el fracaso de la solución propuesta.

2. Caso de estudio

El equipo de trabajo de la oficina desarrolla, mantiene y da soporte a muchas aplicaciones de diferentes dominios. La mayoría de estas aplicaciones son implementadas usando los mismos lenguajes, librerías y tecnologías.

Algunas de las aplicaciones del CeSPI incluyen sistemas de manejo de contenidos, sistemas administrativos de uso interno, un liquidador de sueldo y otras aplicaciones al servicio de las necesidades de la UNLP.

Algunas de los desarrollos del CeSPI son sistemas de:

- Manejo de contenidos. De los cuales se administran aproximadamente 30 instancias.
- Gestión de escuelas de pregrado. Se cuenta con 4 instancias.
- Gestión de expedientes.
- Administración de licencias médicas.
- Gestión de becas estudiantiles.
- Administración del Albergue Universitario.
- Administración de la mesa de ayuda.
- Presentación de proyectos de extensión.
- Gestión de bibliotecas. Del mismo existen alrededor de 20 instancias.
- De uso administrativos por CeSPI como el sistema de gestión de auditorías de loterías provinciales.
- Y en la brevedad el liquidador de sueldos de la Universidad que se encuentra actualmente en desarrollo.

Gran parte de los sistemas anteriormente mensionados utilizan un sistema de autenticación centralizado y se integran con la interfaz de programación de aplicaciones (API) de servicios de la UNLP.

El CeSPI mantiene una gran cantidad de aplicaciones, algunas de las cuales están escritas en el lenguaje de programación PHP y otras en el lenguaje Ruby. El gran número de aplicaciones en producción complejizaba su

mantenimiento, e hizo necesario comenzar a utilizar herramientas que permitieran el manejo, instalación y aprovisionamiento de los servidores de forma dinámica, escalable y consistente.

El incentivo del CeSPI por querer mejorar la infraestructura de monitoreo en sus aplicaciones está estrechamente vinculado con su cultura de trabajo. En la sección 2.1 se explica en qué consiste esta forma de trabajo, y de qué manera la aplicación de la cultura DevOps, la estrategia Lean y la Entrega Continua (CD) se vinculan con la necesidad de mejorar su infraestructura de monitoreo de la oficina.

Con el objetivo de contextualizar el trabajo, en la sección 2.2 se describen las herramientas y tecnologías usadas en el CeSPI, se explican las características de la infraestructura donde se despliegan las aplicaciones y los motivos de por qué está implementada de esa manera. Además, se muestran algunas de las particularidades que han sido tenidas en cuenta al diseñar la solución de monitoreo.

Finalmente en la sección 2.3 se hace mención las diferentes técnicas de monitoreo que se utilizan actualmente sobre las aplicaciones y se explican cuales son los objetivos buscados con la implementación de la arquitectura de monitoreo propuesta.

2.1. Cultura de trabajo

La forma de trabajo de la oficina está en constante evolución. Dedica tiempo y esfuerzo en mejorar la calidad de las tareas que se realizan y la eficiencia en la que se construyen los productos de *software*. Hoy en día esta metodología de trabajo se encuentra fuertemente inspirada en DevOps.

DevOps es una cultura de trabajo centrada en la comunicación, colaboración e integración entre desarrolladores de *software* y profesionales de operaciones en tecnologías de información. Su objetivo es ayudar a una organización a producir productos y servicios de *software* de forma rápida.

El CeSPI cuenta con un enfoque de trabajo de entregas continuas: produce versiones funcionales de *software* en cortos y frecuentes ciclos, asegurando que el producto pueda ser desplegado de forma confiable en cualquier momento. Este enfoque ayuda a reducir costos, tiempo y riesgos de cambios en las entregas al permitir más actualizaciones incrementales en aplicaciones en producción.

El equipo de trabajo adquiere cada día más prácticas que tengan un

enfoque Lean. Lean es una manera de abordar el lanzamiento de negocios y productos basada en el aprendizaje validado, experimentación e iteración en los lanzamientos del producto para acortar los ciclos de desarrollo, medir el progreso y ganar valiosa retroalimentación de parte de los clientes.

Uno de los horizontes en cuanto a desarrollo a los que quiere llegar la oficina es que las aplicaciones recorran el circuito *lean startup* conocido como "crear, medir y aprender". Es un proceso iterativo que consiste en transformar ideas en productos, medir el *feedback* generado por los clientes a partir del uso de la aplicación y aprender, a partir del análisis de la información recolectada, para volver a repetir el ciclo.

Esta serie de ciclos comienza con la implementación de un producto mínimo viable. Esto es, la versión de un nuevo producto que permite al equipo recoger con el mínimo esfuerzo la máxima cantidad de conocimiento validado por los consumidores de dicho producto.

La implementación de esta forma de trabajo permite al CeSPI comenzar un proyecto con un gasto relativamente pequeño y ayudar a los programadores a iniciar el proceso de aprendizaje sobre la aplicación de la forma más rápida posible[18, p .2].

Esta forma de trabajo permite que todos los involucrados en el desarrollo del producto conozcan su ciclo de vida completo. Además incentiva la buena documentación de la infraestructura y la disponibilidad de métricas para todos.

La filosofía de trabajo ha cumplido un papel fundamental en la promoción del interés de los programadores y líderes de proyecto de la oficina en llevar a cabo un monitoreo más adecuado de sus productos y servicios.

Esto es en parte así debido a que la medición constante de los datos de las aplicaciones y los sistemas en general, demuestra ser invaluable a la hora de tomar decisiones para contribuir a la mejora continua de dichas aplicaciones y sistemas. La mejora contínua es uno de los pilares de la metodología de trabajo de la oficina, y su relación con el monitoreo es una de las razones principales por las que se decide comenzar este trabajo.

2.2. Tecnologías utilizadas en la oficina

En sus comienzos, la oficina del CeSPI ha utilizado el *framework web* Symfony 1.4 como herramienta de desarrollo. Hace unos años comenzó a

utilizar Rails[6] como principal herramienta de desarrollo. Este *framework* fue elegido por su excelente documentación, activa comunidad, facilidad para desarrollar e implementado en el lenguaje Ruby.

Ruby, como se menciona en su sitio oficial, es un lenguaje creado con el objetivo de ser "el mejor amigo del desarrollador". Es orientado a objetos y su sintaxis es muy amigable en comparación a otros lenguajes de programación. La expresividad del lenguaje permite escribir código conciso, expresivo y fácil de mantener.

Rails cuenta con soporte para las distintas bases de datos que se utilizan en la oficina. Entre ellas se encuentran MySQL, MongoDB, ElasticSearch y Redis.

Inicialmente tanto la administración de los servidores como la instalación de las aplicaciones y actualización de las mismas se realizaba en forma manual. Cuando el número de servidores y aplicaciones comenzó a crecer se analizaron opciones para automatizar su instalación y actualización.

Entre los años 2011 y 2012 se utilizó Capistrano como herramienta de despliegue automático.

Luego de in incidente ocurrido en la oficina donde se rompió un componente de hardware se tuvieron que instalar manualmente todos los servidores con Capistrano. Es por este motivo que se evaluaron herramientas de automatización de la infraestructura y gestión de la misma como código. Es así como hacia el año 2014 se automatizó el 100 % de la infraestructura y despliegue utilizando Chef.

Con Chef se definieron 3 ambientes:

- **Testing**: es el ambiente donde se publica el software en fase de pruebas para que sea probado por un grupo definido de personas, entre las que se incluye el usuario final o representantes del mismo.
- **Pre-producción**: es la instancia previa a producción, y consiste en un ambiente replicado e idéntico al productivo. En este entorno se verifica el correcto funcionamiento de la aplicación y se realizan los ajustes necesarios en caso de no ser así, evitando que los problemas se descubran en el pasaje a producción.
- **Producción**: es el ambiente que tiene todos los servicios productivos.

[6]Cf. `http://rubyonrails.org/`

Este ambiente cuenta con políticas estrictas en cuanto al acceso y la administración del mismo.

El ambiente de *Testing* a diferencia del de Producción y Pre-producción aún requería el uso de Capistrano por parte de los desarrolladores para realizar el despliegue de las aplicaciones, lo cual hacía que el ambiente de *Testing* no sea exactamente igual al del resto.

Con el objetivo de minimizar las diferencias entre los ambientes, la oficina comenzó la evaluación de la utilización de contenedores Docker, transición que finalizó de forma integra a comienzos del 2017.

Docker es un proyecto de código abierto para la automatización del despliegue de aplicaciones dentro de contenedores de *software* en Linux y Windows. Los contenedores proporcionan una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo.

Docker facilita la construcción de una infraestructura dinámica, escalable y tolerante a fallos que permite el despliegue seguro y automático de aplicaciones, al mismo tiempo que reduce la brecha entre desarrolladores de *software* y profesionales de operaciones en tecnologías de información.

Para lograr que la infraestructura sea realmente escalable y tolerante a fallos es necesario utilizar un *cluster* de Docker, como por ejemplo: Docker Swarm, Apache Mesos, Kubernetes y Rancher Cattle.

Se ha decidido emplear Rancher ya que permite crear los distintos ambientes y definir sobre cada uno de ellos la tecnología de clusterización que se utilizará.

Rancher es una plataforma de código abierto que facilita la orquestación, disponibilidad, configuración y enlace de contenedores. Esta herramienta permite resolver gran parte de los desafíos críticos presentes al ejecutar aplicaciones en contenedores de Docker.

Rancher provee un conjunto completo de servicios para los contenedores que corren en él, simplificando el *discovery* de servicios, *scheduling*, *health checking*, despliegue y el escalado de los mismos.

2.3. Objetivo de la implementación

En este contexto es importante construir una infraestructura de monitoreo que permita obtener valiosa información sobre las reglas de negocio de las aplicaciones que se desarrollan en la oficina, de forma que permita al

equipo de trabajo contar con datos duros para tomar decisiones de negocio más confiables.

Además, los desarrolladores del CeSPI manifiestan que sería útil contar con una herramienta de recolección y análisis de información sobre el rendimiento de las aplicaciones. Esta herramienta podría ayudar a encontrar puntos débiles y cuellos de botella en las aplicaciones.

A las personas encargadas de mantener la infraestructura de la oficina les interesa tener un seguimiento de datos de uso del sistema operativo y las *hardware*, como el uso de CPU, la cantidad de memoria utilizada y disponible, número de tareas en espera, uso de la memoria swap, cantidad de procesos, espacio utilizado en disco.

Anteriormente los desarrolladores no contaban con una implementación formal de monitoreo, sino que recurrían a diversas técnicas para llevar a cabo el seguimiento de las aplicaciones, detectar errores y prevenir su aparición.

Una de las técnicas utilizadas por el grupo de desarrollo era la búsqueda y lectura de registros de *logs* de forma manual. El programador se conectaba a través de SSH al servidor de la aplicación y visualizaba los *logs* directamente mediante la consola.

Si el desarrollador requería conocer información en tiempo real de los recursos del servidor donde estaban corriendo las aplicaciones, éste debía conectarse al servidor y ver la información utilizando comandos como: `htop`, `atsar`,`iostat`, `vmstat`, `df` y `uptime`.

Como las aplicaciones en la oficina utilizan mayormente el motor de bases de datos MySQL para obtener una devolución del funcionamiento de las consultas se utilizaba una herramienta que permitía configurar el envío de reportes a través de correo electrónico usando la funcionalidad `slow_query_log`. Esta funcionalidad guarda información de consultas que toman mucho tiempo en llevarse a cabo. Este tiempo es por defecto de 10 segundos, pero puede ser definido por un usuario.

Para ser notificados de errores en producción de las aplicaciones Rails, los programadores hacían uso de la gema *Exception Notifier*[7]. Esta librería permite configurar en el código de la aplicación, una dirección de correo para recibir las notificaciones de errores en la aplicación, y se acciona cuando la aplicación devuelve a un usuario un código HTTP de error.

[7]Cf. `https://github.com/smartinez87/exception_notification`

La notificación cuenta con valiosa información del contexto de la aplicación. Algunos ejemplos son el tipo de solicitud, los datos de la sesión del usuario, los parámetros de la petición y también el *stake trace*, que contiene información de las llamadas a los métodos que dieron origen al error.

Para el monitoreo y alerta de incidentres del ambiente de producción de la oficina se utiliza con Nagios.

Nagios es una aplicación libre y de código abierto que monitorea sistemas, redes e infraestructura.

La configuración del monitoreo actual no se encuentra lo suficientemente afinada por lo que se reciben notificaciones por cada instancia de un incidente, sin correlacionar ni establecer dependencias entre las alertas lo cual termina siendo poco útil para los usuarios.

La oficina podría beneficiarse con una infraestructura de monitoreo que brinde información útil a los usuarios, sea fácil de utilizar, pueda adaptarse a una arquitectura de contenedores y sea dinámica y escalable.

El primer paso para implementar la nueva infraestructura de monitoreo será recolectar métricas en tiempo real que provengan del uso de las aplicaciones y los contenedores, y obtener de ellos valiosa información que nos permita prevenir errores y tomar mejores decisiones de diseño.

Luego hacer uso de la información que brindan los *logs* de forma más intensiva. Unificar el formato de los *logs* y centralizar los mismo que provengan de diversas fuentes en un único lugar.

La infraestructura de aplicaciones y servicios que se quiere monitorear es dinámica, escalable, construida de forma automatizada y creada a partir de código reutilizable. Lograr una forma de monitoreo que pueda adaptarse a las características de esta infraestructura es lo que se busca, por lo que habrá que investigar cómo utilizar contenedores de Docker para implementar algunos componentes del sistema de monitoreo.

Con el objetivo de impulsar un sistema que brinde información útil a los usuarios, se aplicarán algunas de las técnicas de visualización efectiva mencionadas en el capítulo anterior (sección 1.5) durante la creación de tableros con gráficos.

Finalmente, se realizará una propuesta de una implementación de alertas de mayor utilidad que evite, en la medida de lo posible, falsos positivos y negativos brindando notificaciones útiles a los usuarios indicando concretamente el problema y/o ayudándolos a prevenirlos.

En los próximos capítulos se expone la forma en que desarrolló una infraestructura de monitoreo que intenta cumplir con las características mencionadas.

3. Almacenamiento de series de tiempo

En este capítulo se explicará cómo se ha logrado recolectar, almacenar y consultar información generada en tiempo real de varias fuentes. Ejemplos de esta información son el uso del disco rígido, los procesadores y la memoria.

En la sección 4.3 se repasarán las características que tienen los datos generados en tiempo real, y se describirán brevemente algunas herramientas que permiten el almacenamiento y consulta de estos datos de forma eficiente. En particular se explicará qué es InfluxDB y cómo usarlo.

En la sección 3.2 se demostrará cómo obtener información valiosa de las aplicaciones Rails a partir de la instrumentación y se dará una introducción a la librería `influxdb-rails`, que permitirá enviar datos tomados de las aplicaciones a la instancia de InfluxDB.

En la sección 3.3 se mostrarán herramientas que nos permitan obtener información del funcionamiento en tiempo real de contenedores Docker. En particular se desarrollará cómo configurar la herramienta cAdvisor para enviar información de los contenedores a InfluxDB.

3.1. Almacenamiento

En la sección 1.2 se han descrito las particularidades que tienen los datos de series de tiempo. Una de estas particularidades es que todas las series de tiempo comparten entre sí una dimensión: la dimensión de tiempo, y por eso son útiles para correlacionar datos de varias fuentes.

Las bases de datos relacionales tradicionales pueden no ser prácticas para manejar y almacenar estos tipos de datos. Una base de datos de series de tiempo es un sistema de *software* que está optimizado para manejarlos de forma correcta, confiable y eficiente.

Se han analizado las herramientas OpenTSDB, InfluxDB y Graphite para almacenar datos de este tipo.

InfluxDB es una base de datos de código abierto implementada en el lenguaje Go con el propósito de manejar datos de series de tiempo con requerimientos de gran disponibilidad y alto rendimiento. InfluxDB no tiene dependencias externas.

InfluxDB viene con una API HTTP integrada, lo que permite que no sea necesario escribir ningún código del lado de servidor para comenzar a

trabajar [7].

Es posible etiquetar los datos, lo que permite una consulta muy flexible. Esta consulta se escribe en un lenguaje similar a SQL, llamado InfluxQL[8].

InfluxDB permite contestar consultas en tiempo real, lo que significa que cada valor se indexa a medida que llega y está disponible casi inmediatamente para su uso.

La versión de código abierto de InfluxDB no permite la configuración de un *cluster* de bases de datos. Para tenerlo es necesario actualizar a la versión comercial de InfluxDB.

Graphite es una herramienta de monitoreo capaz de ser ejecutada en una amplia gama de computadoras, y también en la nube. Graphite es usado para tener un seguimiento del rendimiento de sitios *web*, aplicaciones, servicios de negocio y servidores en una red.

Con Graphite es relativamente sencillo almacenar, recuperar, compartir y visualizar datos de tipo series de tiempo.

Graphite consiste en tres componentes de *software*:

- **carbon:** Un servicio de alto rendimiento que recepta datos de tipo series de tiempo
- **whisper:** Una base de datos sencilla para almacenar datos de tipo series de tiempo
- **graphite-web:** Una interfaz de usuario y API para visualizar gráficos y tableros.

Las métricas son alimentadas a través del servicio carbon, que envía datos a las bases de datos whisper para su almacenamiento a largo plazo. Los usuarios interactúan con la interfaz de graphite-web o con su API, que a su vez consulta a carbon y whisper por los datos necesarios para construir los gráficos requeridos.

La plataforma *web* de Graphite ofrece una variedad de estilos y formatos de salidas[9], incluyendo imágenes, archivos separados por comas, XML y JSON.

---

[8]Documentación oficial: `https://docs.influxdata.com/influxdb/v0.9/query_language/query_syntax/`

[9]Más información: `http://graphite.readthedocs.io/en/latest/overview.html`

OpenTSDB es una base de datos de series de tiempos distribuida y escalable, escrita sobre HBase. OpenTSDB almacena, indexa y sirve métricas recolectadas de sistemas de computadoras, como por ejemplo redes, sistemas operativos y aplicaciones, a gran escala, y hace que estos datos sean accesibles y graficables de forma fácil.

OpenTSDB permite recolectar métricas de hosts y aplicaciones a una taza de tiempo alta y es capaz de recolectar miles de métricas de decenas de miles de hosts y aplicaciones, y almacenar miles de millones de valores estadísticos.

OpenTSDB es *software* libre y está disponible en ambas licencias LGPLv2.1 + GPLv3 [10].

Finalmente se ha elegido InfluxDB para la implementación por sobre las demás herramientas ya que es *open source*, tiene un buen rendimientos para lo que se lo va a utilizar y existen librerías para realizar la conexión con las aplicaciones de la oficina de forma simple. Además, InfluxDB tiene un lenguaje de consulta similar a SQL, lo que podría facilitar el desarrollo de consultas al personal de la oficina.

Para ejecutar InfluxDB una vez instalado, basta con correr el comando `service influxdb start`.

Una vez que InfluxDB está instalado, es posible crear una base de datos, insertar datos y realizar consultas usando la API de InfluxDB. Incluso se puede hacer todo esto desde la consola usando el comando `curl`:

1. Crear una base de datos de nombre monitoreo:

```
curl -XPOST 'http://localhost:8086/query' --data-urlencode "q
    =CREATE DATABASE monitoreo"
```

2. Insertar algunos datos:

```
curl -XPOST 'http://localhost:8086/write?db=monitoreo \
-d 'cpu,host=server01,region=uswest load=42
    1434055562000000000'

curl -XPOST 'http://localhost:8086/write?db=monitoreo \
-d 'cpu,host=server02,region=uswest load=78
    1434055562000000000'

curl -XPOST 'http://localhost:8086/write?db=monitoreo \
-d 'cpu,host=server03,region=useast load=15.4
    1434055562000000000'
```

3. Consultar los datos:

```
curl -G http://localhost:8086/query?pretty=true --data-
    urlencode "db=monitoreo" \
--data-urlencode "q=SELECT * FROM cpu WHERE host='server01'
    AND time < now() - 1d"

curl -G http://localhost:8086/query?pretty=true --data-
    urlencode "db=monitoreo" \
--data-urlencode "q=SELECT mean(load) FROM cpu WHERE region='
    uswest'"
```

InfluxDB utiliza varios puertos de red para funcionar. Todos los mapeos de puertos pueden ser modificados en el archivo de configuración que suele estar localizado en `/etc/influxdb/influxdb.conf`.

Por defecto, InfluxDB usa los siguientes puertos de red:

- El puerto *Transmission Control Protocol* (TCP) **8086** es usado para la comunicación entre cliente y servidor sobre la API HTTP de InfluxDB
- El puerto TCP **8088** es usado para el servicio *Remote Procedure Call* (RPC) de *backup* y restauración.

Además de estos puertos, InfluxDB ofrece múltiples *plugins* que pueden requerir puertos personalizados.

3.2. Recolección de datos de aplicaciones

En esta sección se explicará cómo enviar datos de las aplicaciones *web* a InfluxDB. Para enviar datos se ha usado la gema `influxdb-rails`. Esta gema proporciona métodos para vincular Rails con InfluxDB.

Para que la aplicación Rails pueda hacer uso de estos métodos, es necesario agregar la siguiente línea al archivo `Gemfile`:

```
gem "influxdb-rails"
```

Una vez que se tiene instalada la gema, es posible configurar el adaptador `InfluxDB::Rails` en un archivo de configuración de la aplicación. La configuración por defecto debería verse de la siguiente manera:

```
InfluxDB::Rails.configure do |config|
  config.influxdb_database = "rails"
  config.influxdb_username = "root"
  config.influxdb_password = "root"
  config.influxdb_hosts = ["influx"]
  config.influxdb_port  = 8086
end
```

El código Ruby anterior configura los datos necesarios para que la aplicación pueda comunicarse con la base de datos. Estos datos son el nombre de la base de datos, el nombre de usuario, la contraseña del usuario, el nombre del host y el número de puerto por el cual comunicarse con la API de InfluxDB.

Sin necesidad de configurar nada más, la gema brinda automáticamente reportes de los tiempos de ejecución de cada solicitud *web* para los controladores, las vistas y las bases de datos. Pero además, posibilita la realización de llamadas al cliente InfluxDB directamente, escribiendo datos arbitrarios de la siguiente manera:

```
InfluxDB::Rails.client.write_point "events",
  tags:  { url: "/ejemplo", user_id: 1 },
  values: { value: 0 }
```

A continuación se mostrará un ejemplo concreto de cómo se puede utilizar esta herramienta para generar métricas útiles para el equipo de desarrollo y para los dueños de un sistema:

La UNLP brinda un servicio para que los alumnos obtengan su libreta sanitaria que permite registrar las consultas médicas que realicen y solicitar turnos para ir a los consultorios médicos. Este sistema consiste en una aplicación desarrollada en el *framework* Rails.

A la gerencia le parece importante mantener un seguimiento de los inicios de sesión de usuarios en la aplicación. Es posible utilizar la librería `influxdb-rails` para cumplir este objetivo, escribiendo el siguiente código Ruby para que sea ejecutado cada vez que un usuario inicia sesión de forma exitosa:

```
InfluxDB::Rails.client.write_point(
  "logins",
  tags:  {
    ip: request.remote_ip,
    user_id: current_user.id,
    academic_unit: current_user.academic_unit
  },
```

```
  values: { value: request_time }
)
```

Las expresiones `request.remote_ip` y `current_user.id` contienen respectivamente la dirección IP y el identificador único del usuario que se encuentra iniciando sesión en alguno de los sistemas de la UNLP. La dependencia a la cual pertenece el usuario es obtenida a través de `current_user.academic_unit`. El tiempo quedemoró la petición *web* en resolverse se puede encontrar en la variable `request_time`.

A partir de esta información que se está enviando a InfluxDB, es posible obtener varias métricas que pueden ser útiles para los desarrolladores, profesionales de IT y otros. Algunos ejemplos de estas métricas son:

- tiempo de respuesta promedio para la operación de inicio de sesión
- promedio de accesos por cada dependencia
- accesos de cada dependencia agrupados por franja horaria

Este tipo de información puede ayudar a guiar decisiones dentro de las distintas áreas del CeSPI.

Rails incluye una API de instrumentación [10] que sirve para medir eventos. Con esta API, los desarrolladores pueden elegir ser notificados cuando ciertas acciones ocurren dentro de las aplicaciones.

La API de instrumentación provee enganches, o *hooks* a los cuales los desarrolladores pueden suscribirse y así ser notificados en caso de que determinados eventos se produzcan. De esta forma, se podrán tomar acciones a partir de la ocurrencia de estos eventos. Además, la API provee herramientas para que un programador pueda crear sus propios *hooks* a los cuales subscribirse.

Por ejemplo, existe un *hook* provisto por la librería de Rails, *Active Record*[11], que es llamado cada vez que se lleva a cabo una consulta *Structured Query Language* (SQL) en una base de datos. Es posible subscribirse a este

[10] Guía oficial de instrumentación: `http://guides.rubyonrails.org/active_support_instrumentation.html`

[11] Guía oficial de Active Record: `http://guides.rubyonrails.org/active_record_basics.html`

hook, y utilizarlo para tener un seguimiento del número de consultas realizadas.

Podemos aprovechar las facilidades que provee la API de instrumentación de Rails para enviar a InfluxDB información sobre las consultas que se hacen a la base de datos, agregando las siguientes líneas a un archivo de configuración:

```
ActiveSupport::Notifications.subscribe("sql.active_record") do |
   name, start, finish, id, payload|
  InfluxDB::Rails.client.write_point name,
   value: (finish-start),
   start: start,
   finish: finish,
   name: name
end
```

La gema `influxdb-rails` viene configurada por defecto para apoyarse en el *hook* `process_action.action_controller` y enviar a InfluxDB información sobre los tiempos de respuesta de cada petición *web*, distinguiendo el tiempo que demora en el *renderizado* de las vistas, el procesamiento de los controladores y las consultas a las bases de datos.

Para ejecutar la aplicación Rails usando contenedores de Docker, se agregará el archivo Dockerfile en el directorio raíz de la aplicación:

```
FROM ruby:2.3.1

RUN apt-get update -qq && apt-get install -y build-essential libpq-
   dev nodejs sqlite libsqlite3-dev

RUN mkdir /app

WORKDIR /app

COPY Gemfile Gemfile.lock ./
RUN gem install bundler && bundle install

COPY . ./

EXPOSE 3000

# Configure an entry point, so we don't need to specify
# "bundle exec" for each of our commands.
ENTRYPOINT ["bundle", "exec"]
```

```
CMD ["rails", "server", "-b", "0.0.0.0"]
```

La línea `FROM ruby:2.3.1` declara que el contenedor se basa en la imagen de Docker de nombre `ruby:2.3.1`. La siguiente lìnea se ocupa de instalar las librerías necesarias para usar la aplicación Rails.

Luego se crea la carpeta de la aplicación, se copian los archivos `Gemfile` y `Gemfile.lock`, donde se describen las dependencias de la aplicación, y se instalan las gemas de esos archivos con el comando `bundle install`.

Finalmente, se copia el contenido de la aplicación al contenedor Docker, se expone el puerto 3000, y se corre la aplicación con el comando `rails server -b 0.0.0.0`. En la sección A.3 se explica un poco mas de esta herarmienta.

Se mostrará a continuación cómo se puede hacer para correr la aplicación al mismo tiempo que se ejecutan sus servicios. Para esta tarea se utilizará Compose de Docker. Compose permite definir una configuración de los servicios de una aplicación en un archivo, e iniciar todos estos servicios con un sólo comando.

Se ha escrito una explicación un poco más detallada de Compose en el sección A.4.

El siguiente archivo Docker Compose es un ejemplo de cómo configurar la aplicación Rails para trabajar con InfluxDB.

```
version: "2"
services:
  app:
    build: app
    command: rails server -p 3000 -b '0.0.0.0'
    volumes:
      - ./app/:/app
    ports:
      - "3000:3000"
    networks:
      - metricnet

  influxsrv:
    image: influxdb:1.0.0-rc1
    ports:
      - "8083:8083"
      - "8086:8086"
    expose:
      - "8090"
      - "8099"
```

```
    networks:
      - metricnet

networks:
  metricnet:
    driver: bridge
```

En el archivo se definen dos servicios. La aplicación Rails es el servicio de nombre `app`, y la base de datos InfluxDB es el servicio de nombre `influx`. Ambos utilizan la red `lognet` para comunicarse entre sí.

`app` usa el Dockerfile que se describió anteriormente para inicializar el servicio de la aplicación, y define los puertos externo e interno `3000:3000`.

`influx` utiliza la imagen `influxdb:1.0.0-rc1` y configura algunas variables de ambiente, como lo son el nombre del usuario administrador, la contraseña del usuario y el nombre de la base de datos a crear por defecto. Finalmente define los puertos por donde estará la API y el cliente *web*.

3.3. Recolección de datos de contenedores de Docker

Como se ha mencionado anteriormente, el CeSPI usa contenedores de Docker en su infraestructura. Mientras estos contenedores se encuentran en ejecución, es posible tomar información de su funcionamiento en tiempo real.

Para realizar la recolección de datos de los contenedores, donde se mantienen en ejecución las aplicaciones, se han investigado las herramientas cAdvisor y Telegraf.

cAdvisor es un demonio que recolecta, agrega, procesa y exporta información de contenedores en ejecución. Específicamente, para cada contenedor guarda parámetros de aislamiento de recursos, uso de recursos históricos, histogramas de uso completo de recursos históricos y estadísticas de red. Son exportados estos datos tanto para la máquina host donde está corriendo el demonio como para cada uno de los contenedores en la misma.

cAdvisor facilita a los usuarios el entendimiento del uso de recursos y las características de rendimiento de los contenedores en ejecución. Tiene soporte nativo para contenedores Docker y puede utilizarse con casi cualquier otro tipo de contenedor[12].

[12]Repositorio oficial de cAdvisor: `https://github.com/google/cadvisor`

Telegraf es un agente escrito en el lenguaje de programación Go para recolectar, procesar, agregar y escribir métricas. Fue diseñado con el objetivo de requerir poca memoria, y de proveer un sistema de *plugins* de forma que los desarrolladores de la comunidad pudieran fácilmente agregar soporte para recolección de métricas de diferentes servicios y APIs.

Telegraf utiliza cuatro conceptos distintos de *plugins*:

- **Input Plugins:** Recolectan métricas del sistema, servicios o APIs de terceros.
- **Processor Plugins:** Transforman, decoran y filtran métricas.
- **Aggregator Plugins:** Crean métricas de agregación, como promedio, mínimo o máximo.
- **Output Plugins:** Escriben métricas a varios destinos.

Finalmente se ha elegido la herramienta cAdvisor para la implementación por sobre Telegraf porque la primera cuenta con soporte nativo para contenedores de Docker, lo cual es considerado una gran ventaja.

La recomendación oficial para utilizar cAdvisor es que sea ejecutado desde un contenedor de Docker a partir de la imágen oficial[13]. Con el siguiente comando se puede iniciar la ejecución de la imagen mencionada para monitorizar los recursos de una máquina y todos los contenedores corriendo en la misma:

```
sudo docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:rw \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor:latest
```

Además cAdvisor permite exportar estadísticas a distintos tipos de herramientas de almacenamiento por medio de *plugins*. Las herramientas que tienen integración oficial con cAdvisor son:

[13] `https://hub.docker.com/r/google/cadvisor/`

- BigQuery
- ElasticSearch
- InfluxDB
- Kafka
- Prometheus
- Redis
- StatsD

Además de las herramientas mencionadas, cAdvisor permite la salida de información a salida estándar (STDOUT).

Se utilizará un *plugin* para permitir a cAdvisor enviar la información recolectada a InfluxDB. Para configurar la comunicación con la instancia de InfluxDB que se desea utilizar, se dispone de las siguiente opciones[14]:

```
# Elección de manejador de almacenamiento
-storage_driver=influxdb

# IP y puerto de la base de datos. Por defecto es 'localhost:8086'
-storage_driver_host=ip:port

# Nombre de la base de datos. Por defecto es 'cadvisor'
-storage_driver_db

# Usuario de la base de datos. Por defecto es 'root'
-storage_driver_user

# Contraseña de la base de datos. Por defecto es 'root'
-storage_driver_password

# Indicador de conexión segura con la base de datos. Falso por
    defecto.
-storage_driver_secure
```

Para configurar el servicio de cAdvisor se agregan las siguientes líneas al archivo `docker-compose.yml`:

[14]Guía oficial: `https://github.com/google/cadvisor/blob/master/docs/storage/influxdb.md`

```
cadvisor:
  image: google/cadvisor:v0.24.0
  command: -storage_driver=influxdb \
          -storage_driver_db=cadvisor \
          -storage_driver_host=influxsrv:8086
  ports:
    - "9090:8080"
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
  networks:
    - lognet
```

Con estas configuraciones se logran recolectar los datos de los contenedores Docker y configurar cAdvisor para enviarlos a la base de datos de series de tiempo.

4. Almacenamiento de logs

Para implementar la nueva infraestructura de monitoreo, y con el objetivo de darles uso real a los registros de las aplicaciones, el primer paso será intentar utilizar la información que brindan los *logs* de forma más intensiva.

En Rails, los *logs* son complejos. Cada registro puede ocupar varias líneas, y cada tipo de *log* tiene su propio formato. En la sección 4.1 se explicará cómo extraer valiosa información de los *logs* de las aplicaciones del CeSPI.

Se utilizarán los *logs* de NGINX para recolectar datos precisos sobre la velocidad de respuesta de las aplicaciones ante solicitudes *web*. En la sección 4.2 se explicará cómo configurar la herramienta para que la información recolectada sea útil en un ambiente de trabajo con contenedores de Docker.

Para poder hacer uso de la información de los registros de ambas fuentes, se necesitará algún tipo de herramienta de almacenamiento y búsqueda. En la sección 4.3 se describirá cómo configurar ElasticSearch como motor de búsqueda y análisis de *logs*.

En la sección 4.4 se explicará cómo unificar los *logs* de diversas fuentes y cómo centralizarlos en la base de datos descripta en la capítulo 3. Se utilizará Fluentd para lograr que las aplicaciones y las instancias de NGINX se comuniquen correctamente con ElasticSearch, y se mostrará cómo configurarlo.

Una vez que se tenga todo el sistema de recolección, unificación, centralización y almacenamiento de *logs* configurado, se demostrará cómo obtener información útil a través de una API y utilizando un lenguaje de consultas. En la sección 4.5 se darán algunos ejemplos de consultas que se podrán obtener al sistema en funcionamiento.

4.1. Configuración de las aplicaciones

Para poder hacer uso de los *logs* de las aplicaciones, se debió tener en cuenta que la oficina se encuentra en un proceso de migración de infraestructura a contenedores de Docker.

La manera recomendada en la documentación de Docker para realizar logging, es imprimiendo los *logs* de las aplicaciones en STDOUT [15]. Es por ello que se debió configurar los *logs* de las aplicaciones para que sean escritos en la

[15]Cf. `https://docs.docker.com/engine/admin/logging/view_container_logs/`

salida estándar. Utilizar esta estrategia permite que se cuente con información del contenedor en ejecución, como su identificador y su nombre.

Docker tiene soporte para múltiples mecanismos de logging. Al iniciar el daemon de Docker se puede usar la bandera `--log-opt NAME=VALUE` para especificar el *driver* para realizar *logging* que usará por defecto el contenedor.

Para este trabajo se utilizará otra forma de configurar el *driver*, que es a través de la especificación de la opción `log-driver=VALUE` durante el comando `docker run`[16].

Una vez hecho esto, el siguiente paso es configurar las aplicaciones para que envíen sus *logs* a la salida estándar.

Por defecto Rails mantiene un archivo separado para los *logs* por cada ambiente de ejecución, y para hacer uso de la información de los *logs* utiliza de forma predeterminada la clase `ActiveSupport::Logger`.

En Rails es posible cambiar la configuración de la clase que maneja los *logs* especificando la alternativa en un archivo de configuración de la siguiente manera:

```
config.logger = SomeLogger
```

Para que Docker capture correctamente los *logs* de Rails es posible agregar la siguiente especificación al archivo `config/application.rb` de la aplicación:

```
config.logger = ActiveSupport::Logger.new(STDOUT)
```

Los *logs* de Rails tienen algunas características que hacen que sea difícil trabajar con ellos. En primer lugar, no siguen un formato estándar. Esto hace que para distintos tipos de *logs* sea necesario usar diferentes estrategias para recuperar información valiosa. En segundo lugar, un solo *log* puede ocupar varias líneas, lo que hace complejo identificar dónde termina un *log* y dónde empieza el siguiente.

Para resolver estos problemas se utilizó una gema llamada lograge[17]. Esta gema se encarga de transformar complejos *logs* multilínea de Rails en simples *logs* con datos importantes bien identificados de una sola línea.

---

[16] `https://docs.docker.com/engine/admin/logging/overview/`

[17] Cf. `https://github.com/roidrage/lograge`

Por ejemplo, un *log* de Rails, creado al acceder a la página *home* de una de las aplicaciones del CeSPI se ve de la siguiente manera:

```
Started GET "/" for 126.0.0.1 at 2012-03-10 14:28:14 +0100
Processing by HomeController#index as HTML
  Rendered text template within layouts/application (0.0ms)
  Rendered layouts/_assets.html.erb (2.0ms)
  Rendered layouts/_top.html.erb (2.6ms)
  Rendered layouts/_about.html.erb (0.3ms)
  Rendered layouts/_google_analytics.html.erb (0.4ms)
Completed 200 OK in 79ms (Views: 78.8ms | ActiveRecord: 0.0ms)
```

Usando lograge, el mismo *log* es transformado en:

```
method=GET path=/ format=html controller=HomeController action=
    index status=200
duration=79.0 view=78.8 db=0.0
```

Para configurar lograge en Rails, debe incluirse la gema con el mismo nombre a la lista de dependencias de la aplicación. Para hacerlo basta con agregar la siguiente línea de código al archivo `Gemfile`:

```
gem 'lograge'
```

Luego debe agregarse en el archivo `config/application.rb` la siguiente especificación:

```
class Application < Rails::Application
  config.logger = ActiveSupport::Logger.new(STDOUT)
  config.log_formatter = nil
  config.log_level = :info

  config.lograge.enabled = true
  config.lograge.formatter = Lograge::Formatters::Json.new
  config.lograge.custom_options = lambda do |event|
    {
      app_timestamp: event.time
    }
  end
end
```

Con estas configuraciones ya se cuenta con una aplicación Rails que envía la información de sus *logs* a la salida estándar en un formato cuyo procesamiento se puede realizar de forma más sencilla.

4.2. Configuración de Nginx

NGINX es una herramienta multiplataforma que funciona como servidor *web* o proxy inverso ligero[18] de alto rendimiento. Es *software* libre y de código abierto, y está licenciado bajo la licencia *Berkeley Software Distribution* (BSD) simplificada.

Es posible usar NGINX como un balanceador de carga HTTP para distribuir tráfico entre varios servidores de aplicaciones y para aumentar el rendimiento, escalabilidad y confiabilidad de aplicaciones *web* [19].

NGINX es conocido por su alta performance, estabilidad, rico conjunto de utilidades, configuración sencilla y bajo consumo de recursos.

La infraestructura de la oficina a partir de contenedores de Docker, nos otorga casi gratuitamente la funcionalidad de balanceo de carga. En este trabajo se usará NGINX simplemente como un medio confiable y eficiente para obtener información de todas las solicitudes que reciban las aplicaciones *web*.

De esta forma, se podrá obtener información precisa sobre el tiempo de respuesta de las aplicaciones desde el momento en que se recibe la petición hasta que la solicitud es respondida.

NGINX escribe información sobre solicitudes de clientes en el ***access log*** luego de que la petición *web* es procesada. Por defecto, este archivo se ubica en la ruta `logs/access.log`, y la información es impresa en el mismo en un formato predefinido.

Para sobrescribir la configuración por defecto, se puede usar la directiva `log_format`, que modifica el formato de los mensajes registrados así como la directiva `access_log`, que especifica la ruta del archivo de *logs* y el formato de los mismos.

---

[18]Cf. `https://www.nginx.com/resources/admin-guide/reverse-proxy/`

[19]Cf. `http://nginx.org/en/docs/http/load_balancing.html`

Para ejecutar NGINX se utilizará la imagen de Docker oficial [20] de la herramienta, que se encuentra publicada en DockerHub.

Dado que la propuesta fue utilizar Docker como entorno virtual en donde corren los distintos servicios, se ha debido tener en cuenta las particularidades que tiene esta herramienta para poder obtener los *logs* de la aplicación en ejecución.

Al igual que en el caso de las aplicaciones Rails, para obtener los registros del servicio NGINX que se encuentra en ejecución dentro de un contenedor Docker, es necesario hacer uso de uno de sus *drivers*. También en este caso se utilizará la herramienta Fluentd para recolectar los *logs*, por lo que se usará el driver que lleva el mismo nombre.

Por defecto, la imagen de NGINX es configurada para enviar los *logs* principales de acceso y error de NGINX al recolector de *logs* de Docker. Esto se logra enlazando los *logs* con STDOUT y error estándar (STDERR).

En esta sección se han mostrado las configuraciones necesarias para obtener información de los *logs* de NGINX, de forma de que estén disponibles para su recolección por otra herramienta.

4.3. Almacenamiento

Para poder almacenar los *logs* de una forma centralizada, y hacer consultas sobre los mismos de una forma sencilla, se ha elegido utilizar ElasticSearch.

ElasticSearch es un servidor que provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi-tenencia con una interfaz *web* RESTful y con documentos JSON. ElasticSearch fue desarrollado en Java y se encuentra publicado como código abierto bajo las condiciones de la licencia Apache.

ElasticSearch permite almacenar, buscar y analizar datos estructurados y no estructurados, incluyendo texto, *logs* del sistema, registros de base de datos y más. Además puede ser usado para guardar métricas de series de tiempo proporcionadas por otras herramientas.

ElasticSearch permite escalabilidad y posibilidad de agregar nuevas métricas en tiempo de ejecución, tiene facilidades para la integración con APIs

[20] `https://hub.docker.com/_/nginx/`

y herramientas como Logstash, y consigue un buen rendimiento incluso al procesar grandes volúmenes de datos.

Además tiene excelente integración con la herramienta de visualización de datos Kibana, que permite analizar datos de múltiples fuentes y correlacionar métricas almacenadas en ElasticSearch.

ElasticSearch no es una base de datos relacional. Es orientada a documentos, y tiene su propia terminología. Para facilitar la comprensión de su funcionamiento, es importante que se mencionen los conceptos de documento, índice y tipo en ElasticSearch:

- Un **documento** es una unidad básica de información que puede ser indexada. Por ejemplo, se puede tener un documento por un único cliente, otro documento por un producto específico y otro documento por una orden de compras. Estos documentos son expresados en formato JSON.
- Un **índice** es una colección de documentos que tienen características similares. Por ejemplo, es posible tener un índice para datos de clientes, otro índice para un catálogo de productos y otro índice para datos de órdenes de compras. Un índice es identificado por un nombre escrito en minúsculas. Este nombre es usado para referirse al índice al realizar operaciones de indexado, búsqueda, actualización y eliminación sobre los documentos en el mismo.
- Un **tipo** en ElasticSearch representa una clase de documentos similares. Un tipo consiste en un nombre y un mapeo que, como el esquema de una base de datos, describe los campos o propiedades que los documentos de ese tipo pueden tener.

ElasticSearch provee una API RESTful JSON sobre HTTP para realizar todas las operaciones importantes. Por defecto, ElasticSearch corre sobre el puerto 9200. Es posible comunicarse con esta API mediante clientes *web*, o también usando el comando `curl` desde una terminal.

ElasticSearch provee clientes oficiales para muchos lenguajes, como Groovy, JavaScript, .NET, PHP, Perl, Python y Ruby, y hay numerosos clientes e integraciones provistos por la comunidad, que pueden ser encontrados en el sitio *web* de ElasticSearch.

Por ejemplo, si se quisiera indexar un documento por cada empleado, de tal forma que cada documento sea de tipo empleado y ese tipo exista en el índice de la biblioteca, bastaría con llamar a la API de la siguiente manera:

```
PUT /biblioteca/empleado/1
{
    "nombre" : "Juan",
    "apellido" : "Gomez",
    "edad" :    25,
    "areas" :   [ "encargado" ]
}
```

Para esta solución, se ha elegido instalar ElasticSearch en un servidor aparte. Esto otorga por lo menos dos ventajas:

En primer lugar, guardar los datos de los *logs* en un entorno diferente al que corren las aplicaciones, permite tener disponibilidad de los datos en caso de que ocurra un error que afecte a la infraestructura completa.

Además, al tener una sola instancia separada, se logra centralizar los datos de los *logs* y se tiene una única fuente que consultar. En una solución centralizada es importante poder identificar la fuente de cada uno de los *logs*.

4.4. Unificación y recolección

El siguiente paso es tomar los datos de ambas fuentes, unificarlos y enviarlos a la instancia en ejecución de ElasticSearch. Para cumplir con esta tarea se utilizará la herramienta Fluentd.

Fluentd es un recolector de datos multiplataforma y de código abierto escrito en Ruby y desarrollado por la compañía *TreasureData*. Esta herramienta permite unificar el consumo y recolección de datos.

Fluentd cuenta con un sistema flexible de *plugins* que permite a la comunidad extender su funcionalidad. Hoy en día cuenta con más de 300 *plugins* que permiten tomar datos de docenas de fuentes diferentes.

Fluentd es muy utilizado debido a su mínimo consumo de memoria, su gran poder de procesamiento de datos, su robustez y su alta disponibilidad.

Además de Fluentd se han analizado las herramientas Logstash, Heka y Splunk:

- **Logstash** es un motor de recolección de datos de código abierto. Logstash puede unificar dinámicamente datos de fuentes dispares y normalizar los datos en destinos de su elección.

Logstash funciona como un canal de procesamiento de datos del lado del servidor, que toma datos de una multitud de fuentes de forma simultánea, los transforma y envía para que sirvan como entrada de otra herramienta.

Con Logstash es posible recolectar *logs* y cualquier tipo de evento. Estos eventos pueden ser transformados a partir de una amplia gama de entradas, filtros y *plugins* de salida.

- **Heka** es una solución de manejo de *logs* de código abierto. Fue diseñada por el equipo de Mozilla y está escrita en el lenguaje de programación Go. Al igual que Logstash, provee un sistema basado en *plugins* para recolectar y procesar *logs*. Además permite dirigir las salidas a una variedad de destinos, incluyendo ElasticSearch. Lamentablemente, Heka no se encuentra actualmente en mantenimiento.

- **Splunk** es una herramienta comercial para buscar, monitorizar y analizar datos de aplicaciones y sistemas a través de una interfaz *web*. Cuenta con soluciones basadas en la nube. Splunk captura, indexa y correlaciona datos en tiempo real y los almacena en un repositorio, desde donde se pueden recuperar para generar gráficos, alertas y tableros de control definibles por un usuario.

Se ha descartado el uso de Heka, por no ser mantenida de forma activa, y de Splunk, por ser *software* privativo.

Fluentd y Logstash resuelven problemas similares y tienen una fácil integración con Docker. Finalmente se ha elegido Fluentd para esta solución en lugar de Logstash, porque Fluentd se configura usando Ruby, lo que es una gran ventaja para el CeSPI, ya que sus programadores cuentan con experiencia en ese lenguaje. Además, Fluentd cuenta con un enfoque declarativo por lo que su configuración es más sencilla.

Fluentd usa el formato JSON para estructurar los datos, lo que permite unificar todas las facetas del procesamiento de datos de *logs*: recolección, filtrado, *buffering* y envío de *logs* a través de múltiples fuentes y destinos.

Una vez que se tiene Fluentd instalado, de debe configurarlo para que pueda leer los datos de los *logs* desde la salida estándar de Docker, y enviarlos a la instancia de ElasticSearch.

Para lograr este objetivo se agregarán dos gemas a la configuración de Fluentd que permitirán enviar los datos a ElasticSearch y *parsear* los datos de distintas fuentes para tomar la información que se crea importante. Esto se hará en un archivo llamado `start.sh`:

```
#!/bin/sh

gem install fluent-plugin-elasticsearch
gem install fluent-plugin-parser
exec fluentd -c /fluentd/etc/$FLUENTD_CONF -p /fluentd/plugins
    $FLUENTD_OPT
```

Este archivo instala las gemas necesarias, y luego inicia el servicio de Fluentd, definiendo la ruta del archivo de configuración y la carpeta en la que se encuentran los *plugins*.

La gema `fluent-plugin-elasticsearch` permite que Fluentd envíe sus datos a ElasticSearch. Este *plugin* crea índices de ElasticSearch de forma automática. Basta con agregar a la configuración de Fluentd, el atributo `type` con valor ElasticSearch y de forma opcional, parámetros adicionales. Por ejemplo:

```
<match my.logs>
  type elasticsearch
  host localhost
  port 9200
  index_name fluentd
  type_name fluentd
</match>
```

La gema `fluent-plugin-parser` permite a Fluentd reconocer patrones en las cadenas de texto y remitirlas. Por ejemplo, el siguiente código es capaz de reconocer a partir de una expresión regular el *host*, usuario y hora en un mensaje de Apache:

```
<match raw.apache.common.*>
  @type parser
  remove_prefix raw
  format /^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\])$
      /
  time_format \%d/\%b/\%Y:\%H:\%M:\%S \%z
  key_name message
</match>
```

El siguiente paso es escribir la configuración de Fluentd, en el archivo `fluentd.conf`:

```
<source>
  type forward
  port 24224
</source>

<match rails.docker.**>
  type parser
  key_name log
  format json
  reserve_data yes
  add_prefix logs
</match>

<match nginx.docker.**>
  type parser
  key_name log
  format /^(?<remote_addr>[^ ]*) - (?<remote_user>[^ ]*) \[(?<time
      >[^\]]*)\] "(?<request_type>[^ ]*) (?<request_url>[^ ]*) (?<
      request_http_protocol>[^ ]*)" (?<status_code>[^ ]*) (?<
      request_size>[^ ]*) "(?<referer>[^\"]*)" "(?<user_agent
      >[^\"]*)" "(?<unknown>[^\"]*)"?$/
  time_format \%d/\%b/\%Y:\%H:\%M:\%S \%z
  reserve_data yes
  add_prefix logs
</match>

<match logs.rails.**>
  type elasticsearch
  host elasticsearch
  logstash_format true
  logstash_prefix rails-logs
  flush_interval 3s
</match>

<match logs.nginx.**>
  type elasticsearch
  host elasticsearch
  logstash_format true
  logstash_prefix nginx-logs
  flush_interval 3s
</match>
```

A continuación se explicará el funcionamiento del código, bloque por

bloque.

```
<source>
  type forward
  port 24224
</source>
```

Las fuentes de entrada de Fluentd son habilitadas seleccionando y configurando los *plugins* de entrada deseados usando la directiva `source`. La instrucción `type forward` convierte a Fluentd en un *endpoint* TCP. La línea `port 24224`, indica el puerto en donde Fluentd aceptará paquetes TCP [21].

```
<match rails.docker.**>
  type parser
  key_name log
  format json
  reserve_data yes
  add_prefix logs
</match>
```

La directiva `match` busca eventos con etiquetas que coincidan con un patrón y los procesa. En este caso, sólo tomará las entradas con etiquetas que comiencen con `'rails.docker.'`. Las instrucciones `type parser`, `key_name log` y `format json` indican que la clave de nombre `log` en la entrada será analizada sintácticamente y transformada a formato JSON.

La línea `reserve_data yes` es usada para mantener los datos originales. Por ejemplo, si la entrada es la cadena de texto `'log': {üser":1,"num":2}`, la salida es:

```
{"log":"{\"user\":1,\"num\":2}","log.user":1, "log.num":2}
```

Finalmente, la instrucción `add_prefix logs` es utilizada para agregar la subcadena `logs.` al comienzo del nombre de la etiqueta. Por ejemplo, si la etiqueta se llama `rails.docker.app1`, la nueva etiqueta será `logs.rails.docker.app1`.

De esta forma, la salida de este bloque se podrá convertir en la entrada de otro bloque.

---

[21] Cf. `http://docs.fluentd.org/v0.12/articles/config-file`

```
<match nginx.docker.**>
  type parser
  key_name log
  format /^(?<remote_addr>[^ ]*) - (?<remote_user>[^ ]*) \[(?<time
      >[^\]]*)\] "(?<request_type>[^ ]*) (?<request_url>[^ ]*) (?<
      request_http_protocol>[^ ]*)" (?<status_code>[^ ]*) (?<
      request_size>[^ ]*) "(?<referer>[^\"]*)" "(?<user_agent
      >[^\"]*)" "(?<unknown>[^\"]*)"?$/
  time_format \%d/\%b/\%Y:\%H:\%M:\%S \%z
  reserve_data yes
  add_prefix logs
</match>
```

Este bloque funciona de forma similar al anterior. Toma aquellas entradas con etiquetas que comienzan con la cadena `'nginx.docker.''`, es decir, aquellas que provengan de NGINX.

En lugar de utilizar el formato JSON, se usa una expresión regular para recuperar los valores importantes de la expresión. Algunos de estos valores son la marca de tiempo, el tipo de solicitud *web*, la *Uniform Resource Locator* (URL) a la que se intentó acceder, el protocolo HTTP, el código de estado HTTP, el tamaño del mensaje de solicitud y el cliente *web* utilizado.

Las marcas de tiempo suelen ser problemáticas en los *logs*. Cada fuente de registros puede tener formatos de hora y fecha diferentes, lo que dificulta el análisis de estos datos. La instrucción `time_format` permite definir el formato que se quiere que cumplan las fechas y de esa forma evitar estos problemas.

```
<match logs.rails.**>
  type elasticsearch
  host elasticsearch
  logstash_format true
  logstash_prefix rails-logs
  flush_interval 3s
</match>

<match logs.nginx.**>
  type elasticsearch
  host elasticsearch
  logstash_format true
  logstash_prefix nginx-logs
  flush_interval 3s
```

```
</match>
```

Los últimos dos bloques toman como entrada las salidas de los bloques anteriores, y las envían a ElasticSearch con un formato adecuado, haciendo uso del *plugin* `fluentd-plugin-elasticsearch`. Este *plugin* además, crea índices en ElasticSearch.

Teniendo en consideración la arquitectura general de la infraestructura, se ha decidido, para esta solución, que Fluentd sea ejecutado sobre su propio contenedor de Docker. Para lograr que este contenedor se comunique con el contenedor de la aplicación y con la base de datos, se usará la herramienta Compose de Docker.

Para esto, se crea el archivo `docker-compose.yml` y se agrega la siguiente configuración:

```
fluentd:
    image: fluent/fluentd:latest
    ports:
      - "24224:24224"
    volumes:
      - ./fluentd/etc:/fluentd/etc
    command: /fluentd/etc/start.sh
    networks:
      - lognet
```

La línea `image: fluent/fluentd:latest` indica la imagen de Docker que se utilizará. En este caso, se utiliza la imagen oficial de Fluentd[22].

La clave `ports` se utiliza para exponer puertos. Gracias a esta capacidad es posible conectar el contenedor con el sistema externo, para intercambiar información por *sockets*, y de la misma forma, se permite conectar varios contenedores entre sí.

Para usar esta configuración se requiere indicar puerto origen y puerto destino. El puerto origen es interno al contenedor, y el puerto destino existe sólo en el entorno no virtualizado. En el ejemplo, un contenedor abre el puerto interno 24224 y lo mapea con el puerto del entorno externo 24224.

La configuración `volumes` es usada para compartir el directorio `fluentd /etc` con el contenedor. La instrucción command: `/fluentd/etc/start.sh`

[22]Cf. `https://hub.docker.com/r/fluent/fluentd/l`

sirve para que el script que corre Fluentd sea ejecutado al poner en marcha el contenedor.

Finalmente, para que los contenedores con NGINX y con las aplicaciones Rails puedan enviar los datos a Fluentd, se agregan al archivo `docker-compose.yml` las siguientes líneas:

```
nginx:
  logging:
    driver: fluentd

app:
  logging:
    driver: fluentd
```

Se puede encontrar la configuración completa de ejemplo en el anexo C.

Ya se tiene configurada la salida de los *logs* de una aplicación Rails, la recolección y *parseo* de los mismos con Fluentd y el envío de los mismos a la base de datos destinada a este fin con ElasticSearch. Ya es posible consultar a la base de datos información sobre los *logs*.

4.5. Consultas

Una vez que se tiene todo configurado, se puede comenzar a realizar consultas sobre ElasticSearch. A través del comando `curl` se pueden solicitar los datos de los índices creados en la base de datos:

```
curl localhost:9200/_cat/indices?v
```

```
health status index            uuid                   pri rep docs.
   count docs.deleted store.size pri.store.size
yellow open nginx-logs-2017.03.22 fNCeXckpQwyWtr5dntS9aQ 5 1     4
            0      51kb         51kb
yellow open rails-logs-2017.03.22 nh9ID9o3TwmrxYmBOven8Q 5 1    46
            0    87.3kb       87.3kb
yellow open .kibana            sHMjKRIWRuKrtZlZ9b31IQ 1 1         2
            0    10.2kb       10.2kb
```

En la respuesta de la consulta se pueden visualizar tres índices: `.kibana`, rails-logs-2017.02.21 y `nginx-logs-2017.02.21`.

Si se quisiera visualizar la configuración de uno de estos índices, alcanzaría con hacer una consulta como la siguiente:

```
curl localhost:9200/rails-logs-2017.02.21
```


```
{
  "rails-logs-2017.03.22": {
    "aliases": {},
      "mappings": {
        "fluentd": {
          "properties": {
            "@timestamp": {
              "type": "date"
            },
            "action": {
              "fields": {
                "keyword": {
                  "ignore_above": 256,
                  "type": "keyword"
                }
              },
              "type": "text"
            },
            "app_timestamp": {
              "fields": {
                "keyword": {
                  "ignore_above": 256,
                  "type": "keyword"
                }
              },
              "type": "text"
            },
            "container_id": {
              "fields": {
                "keyword": {
                  "ignore_above": 256,
                  "type": "keyword"
                }
              },
              "type": "text"
            },
            "controller": {
```

```
  "fields": {
    "keyword": {
      "ignore_above": 256,
      "type": "keyword"
    }
  },
  "type": "text"
},
"db": {
  "type": "float"
},
"log": {
  "fields": {
    "keyword": {
      "ignore_above": 256,
      "type": "keyword"
    }
  },
  "type": "text"
},
"method": {
  "fields": {
    "keyword": {
      "ignore_above": 256,
      "type": "keyword"
    }
  },
  "type": "text"
},
"path": {
  "fields": {
    "keyword": {
      "ignore_above": 256,
      "type": "keyword"
    }
  },
  "type": "text"
},
"status": {
  "type": "long"
},
"view": {
  "type": "float"
```

```
          },
          ... # más atributos
        }
      }
    },
    ... # más atributos
  }
}
```

En el resultado de la consulta es posible ver la estructura de los registros y la composición del índice que almacena los *logs* de las aplicaciones Rails.

Para visualizar todos los documentos dentro de un índice, se puede ejecutar la siguiente instrucción:

```
curl localhost:9200/nginx-logs-2017.02.21/_search
```


```
{
   "_shards": {
      "failed": 0,
      "successful": 5,
      "total": 5
   },
   "hits": {
      "hits": [
         {
            "_id": "AVpiotVhd2D1TjttLYvK",
            "_index": "nginx-logs-2017.02.21",
            "_score": 1.0,
            "_source": {
               "@timestamp": "2017-02-21T21:45:20+00:00",
               "container_id": "e7e13c19bf035...240b",
               "container_name": "/logger_nginx_1",
               "log": "172.23.0.1 - - [21/Feb/2017:21:45:20
                  +0000] \"GET / HTTP/1.1\" 304 0 \"-\" \"Mozilla
                  /5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
                  KHTML, like Gecko) Chrome/56.0.2924.87 Safari
                  /537.36\" \"-\"",
               "referer": "-",
               "remote_addr": "172.23.0.1",
               "remote_user": "-",
               "request_http_protocol": "HTTP/1.1",
               "request_size": "0",
```

```
                "request_type": "GET",
                "request_url": "/",
                "source": "stdout",
                "status_code": "304",
                "user_agent": "Mozilla/5.0 (X11; Linux x86_64)
                    AppleWebKit/537.36 (KHTML, like Gecko) Chrome
                    /56.0.2924.87 Safari/537.36"
            },
            "_type": "fluentd"
         },
         .
         .
         .
      ],
      "max_score": 1.0,
      "total": 10
   },
   "timed_out": false,
   "took": 2
}
```


Esta consulta retorna información sobre todos los documentos en el índice de requerimientos *web* que han pasado por NGINX. Entre otros datos, es posible ver el identificador y nombre del contenedor, el código de estado HTTP, el cliente *web* utilizado y la marca de tiempo.

Fluentd ha permitido analizar el contenido del mensaje de los *logs* y separar aquellos campos importantes. Esto facilitará la realización de búsquedas en el futuro.

A lo largo de este capítulo se ha conseguido configurar varias herramientas para unificar los *logs* de diferentes fuentes en una base de datos centralizada, y utilizar un motor de búsquedas para obtener información importante de ellos.

En el siguiente capítulo se abordará el tema de la visualización de datos.

5. Visualización

La visualización de datos es el proceso de interpretación, contrastación y comparación de datos que permite un conocimiento en profundidad y detalle de los mismos de tal forma que se transformen en información comprensible para las personas.

En la sección 5.1 se explica qué son las herramientas de visualización de datos y se describen las herramientas Kibana y Grafana.

En la sección 5.2 se explica como configurar Kibana para realizar consultas en nuestra instancia de ElasticSearch. Además se muestra cómo crear gráficos a partir de estos datos y de qué forma explorar los *logs* desde el cliente *web* de Kibana.

En la sección 5.3 se muestra cómo configurar Grafana para que trabaje de forma correcta con InfluxDB. Además se configura un datasource vinculado con la instancia de InfluxDB y se crean tableros con gráficos que hagan uso de esta información.

5.1. Herramientas

Las herramientas de *software* de visualización de datos son aquellas que se ocupan de mostrar datos de distintas fuentes en forma de gráficos, de forma que los usuarios puedan descubrir patrones y entender la información de forma sencilla.

Existen varias herramientas informáticas para visualizar datos. En esta sección se describirán brevemente las herramientas Kibana y Grafana.

- **Kibana** es una plataforma de análisis y visualización de código abierto diseñada para trabajar con ElasticSearch. Se lo puede usar para buscar, observar e interactuar con datos almacenados en índices de esta herramienta.

  Con Kibana es posible realizar análisis de datos complejos y visualizar datos en una variedad de gráficos, tablas y mapas. Figura 6

  Kibana cuenta con una interfaz basada en el navegador, que permite crear y compartir tableros de control dinámicos que muestran cambios en tiempo real a partir de consultas a ElasticSearch.

- **Grafana** es un tablero y componedor de gráficos de propósito general y de código abierto que corre como una aplicación *web*. Es comúnmente

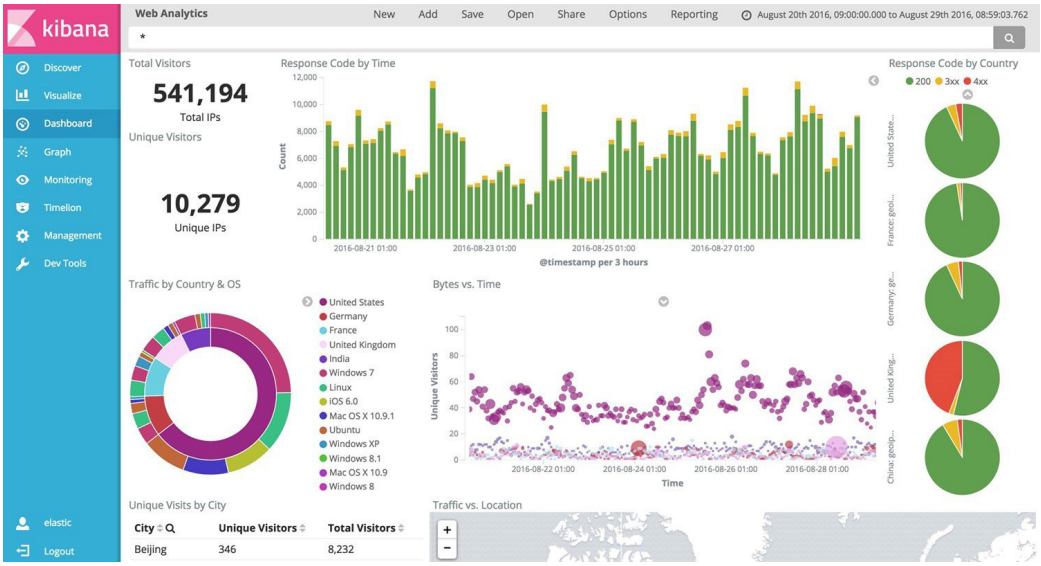

Figura 6: Tablero de Kibana

usado para visualizar datos de series de tiempo para infraestructuras en la *web* y analíticas de aplicación, pero también es usado en sensores industriales, automatización de viviendas, medición del clima y control de procesos. Figura 7

Grafana permite una sencilla extensibilidad y variedad de paneles, con ricas opciones de visualización, y tiene soporte para las fuentes de datos de series de tiempo más populares, incluyendo InfluxDB y ElasticSearch.

Luego de analizar estas opciones, se consensuó que la herramienta de visualización más completa entre ambas era Grafana.

Pero Kibana es un *frontend* de ElasticSearch y está preparado especialmente para realizar consultas sobre esta base de datos de forma simple. Con Kibana es posible hacer consultas directas sobre los registros de *logs* tal cual fueron recuperados e indexados.

Es por esto que se decidió usar ambas herramientas: Kibana para consultar las base de datos de ElasticSearch, y hacer consultas acerca de los *logs*, y Grafana para comunicarse con la instancia de InfluxDB y visualizar datos generados en tiempo real.

Estas herramientas están en constante crecimiento, por lo que no se descarta que en un futuro Grafana incorpore funcionalidades que se encuentran presentes en Kibana, y viceversa.

Figura 7: Tablero de Grafana

5.2. Configuración de Kibana

En el capítulo 4 se ha explicado la forma en que se ha usado ElasticSearch para almacenar los datos obtenidos de *logs*. El siguiente paso consiste en usar Kibana para poder visualizar los datos almacenados en ElasticSearch.

Se usará Kibana a través de la imagen de Docker oficial. Esta imagen provee varios métodos para configurar Kibana. El enfoque convencional es proveer un archivo `kibana.yml`, pero también es posible utilizar variables de ambiente para definir las opciones.

Para hacer esto, se pueden agregar las siguientes líneas al archivo `docker-compose.yml`, definiendo la variable de ambiente `ELASTICSEARCH_URL`:

```
services:
  ...
  kibana:
    image: kibana
    restart: always
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_URL=http://elasticsearch:9200
    networks:
      - lognet
```

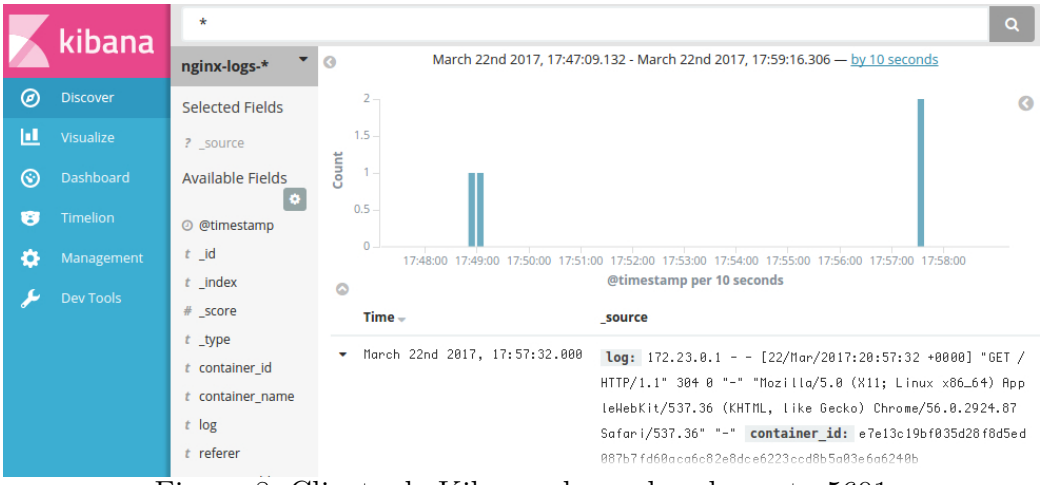

Figura 8: Cliente de Kibana al acceder al puerto 5601

Una vez que está corriendo el contenedor de Kibana, es posible acceder al cliente *web* (Figura 8) a través del puerto 5601. Todo lo que uno necesita es apuntar el navegador *web* a la máquina en la cual Kibana está siendo ejecutado y especificar el número de puerto.

Cuando se accede a Kibana, la página de descubrimiento se carga de forma automática con el patrón del índice por defecto seleccionado. El filtro de tiempo es fijado a los últimos 15 minutos y la consulta de búsqueda es configurada para traer todos los resultados.

Es posible ver el estado de página del servidor de Kibana navegando a la dirección `localhost:5601/status`. La página de estado (Figura 9) muestra información acerca del uso de recursos del servidor y lista los *plugins* instalados.

Para poder comenzar a usar Kibana, es necesario indicar los índices de ElasticSearch que se desean explorar. La primera vez que se accede a Kibana, le es solicitado al usuario definir un patrón que coincida con los nombres de uno o más índices. Una vez hecho esto ya es posible empezar a explorar datos. En la Figura 10 detallamos las partes de la interfaz de Kibana.

Usando Kibana se puede realizar la exploración de datos de forma interactiva a través de la página de descubrimiento. Desde esta página se tiene acceso a cada documento en cada índice que coincida con un patrón de nombres de índices seleccionado.

De esta forma, es posible enviar consultas, filtrar los resultados de las búsquedas y visualizar datos de los documentos. Además se pueden ver los números de documentos que coinciden con la consulta de búsqueda y obtener

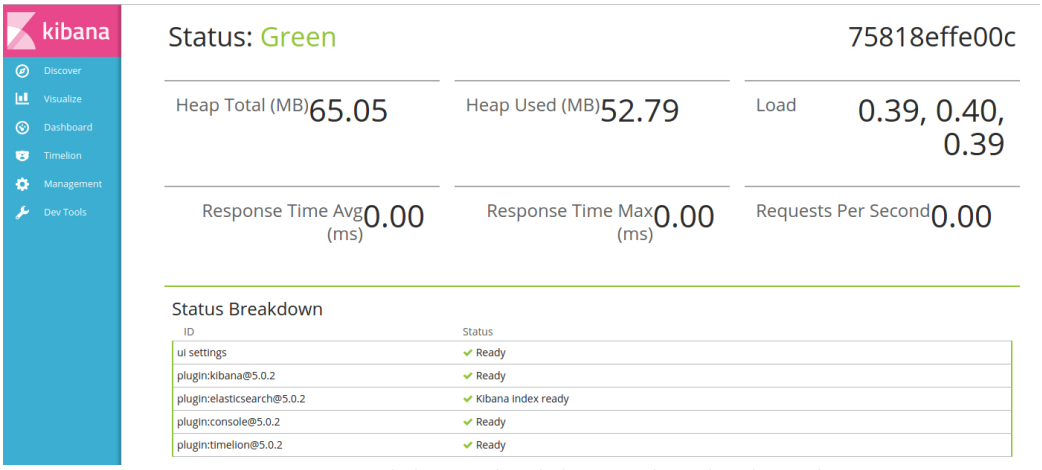

Figura 9: Vista del estado del servidor desde Kibana

estadísticas de valores de los campos.

Si un campo de tiempo es configurado para el patrón de índice seleccionado, la distribución de documentos en el tiempo es mostrada en un histograma en la parte de arriba de la página.

Kibana permite crear visualizaciones de datos en los índices de ElasticSearch, y con ellas construir tableros que muestren gráficos relacionados entre sí.

Las visualizaciones de Kibana están basadas en consultas de ElasticSearch. Al usar una serie de agregaciones de ElasticSearch para extraer y procesar datos, se pueden crear gráficos que muestren tendencias, picos y caídas, cuyo conocimiento puede ser importante.

Es posible crear visualizaciones de distintos tipos. Algunos de estos tipos son:

- **Gráficos de área:** Para visualizar la contribución total de varias series distintas.
- **Tabla de datos:** Para mostrar datos crudos de una agregación compuesta.
- **Gráfico de líneas:** Para comparar diferentes series.
- **Métrica:** Para mostrar un único número.
- **Gráfico de torta:** Para comparar la contribución de cada fuente.

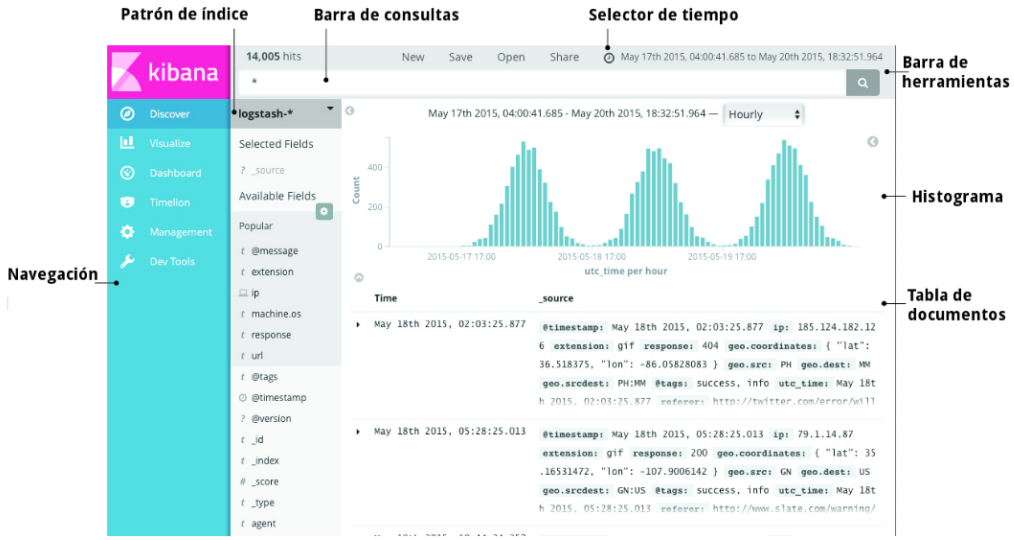

Figura 10: Explicación de las partes de la interfaz de Kibana

- **Nube de etiquetas:** Para mostrar palabras de forma que el tamaño de la palabra corresponda con su importancia.
- **Serie de tiempo:** Para computar y combinar datos de múltiples conjuntos de datos de tipo serie de tiempo.
- **Gráfico de barras vertical:** Para graficar valores y comparar varias dimensiones al mismo tiempo.

El primer paso para crear una visualización es elegir el tipo de gráfico que se quiere mostrar. Luego se debe especificar una consulta de búsqueda. Se puede escribir una consulta nueva o tomar una búsqueda guardada anteriormente. El siguiente paso es elegir la métrica de agregación para la visualización. Ésta puede ser:

- **count** (cantidad)
- **average** (promedio)
- **sum** (suma)
- **min** (mínimo)
- **max** (máximo)
- **unique count** (cantidad de elementos únicos)

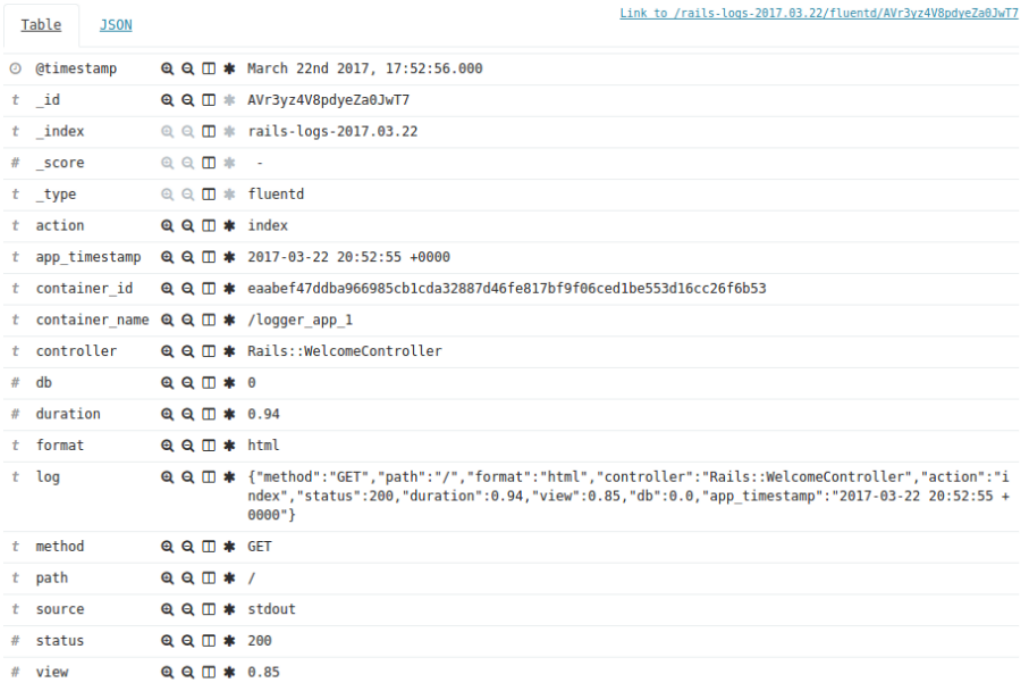

Figura 11: Vista de *logs* de Rails en Kibana

- **median** (mediana o percentil 50)
- **percentiles** (percentiles)
- **percentile ranks** (rangos de percentiles)

Además es posible agrupar los datos por fechas, rangos, términos o filtros. Por ejemplo, si se están indexando *logs* del servidor se puede construir un gráfico de barras que muestre la distribución de solicitudes *web* entrantes por locación geográfica.

En el caso de la infraestructura *web* estudiada, Kibana podría ser extremadamente útil para detectar errores de forma rápida. Ante un error de una aplicación Rails, se generaría un *log* que sería indexado de forma inmediata por la instancia de ElasticSearch.

Si realizamos una búsqueda en Kibana filtrando por tipo de error o por fecha y hora, se encontraría el error en la aplicación de forma rápida y se contaría con toda la información necesaria para reparar el error lo más pronto posible.

En la Figura 11 se puede ver la forma en que se visualiza en Kibana una línea de *logs* tomada de las aplicaciones Rails. En la misma es posible

apreciar cómo se ha logrado separar la información contenida en una única línea de *log* en varios atributos para poder realizar consultas.

En este capítulo se ha mostrado cómo configurar Kibana para generar tableros de control y cómo usarlo para explorar la información de *logs* almacenados en ElasticSearch para detectar errores de forma sencilla.

En la próxima sección se detallará cómo configurar Grafana para visualizar información de la base de datos InfluxDB.

5.3. Configuración de Grafana

Como se ha mencionado anteriormente, se ha elegido Grafana para la creación de los tableros que mostraran la información que se ha recolectado en el capítulo 3.

Para iniciar el servicio de Grafana en Linux, basta con ejecutar el siguiente comando en la terminal:

```
$ sudo service grafana-server start
```

Esta orden inicializa el proceso `grafana-sever` como el usuario Grafana, que fue creado durante la instalación.[23]

Una vez que el proceso está en ejecución es posible acceder al servicio en el puerto 3000 del navegador e iniciar sesión.

Grafana soporta varios *backends* de almacenamiento diferentes, llamados datasources. Cada uno de ellos tiene un editor de consultas visual específico, para facilitar el armado de paneles de control a los usuarios.

Grafana tiene soporte oficial para tomar información de las siguientes bases de datos, herramientas y servicios:

- CloudWatch
- ElasticSearch
- Graphite
- InfluxDB

[23]Información sobre el proceso de instalación de Grafana en `http://docs.grafana.org/`

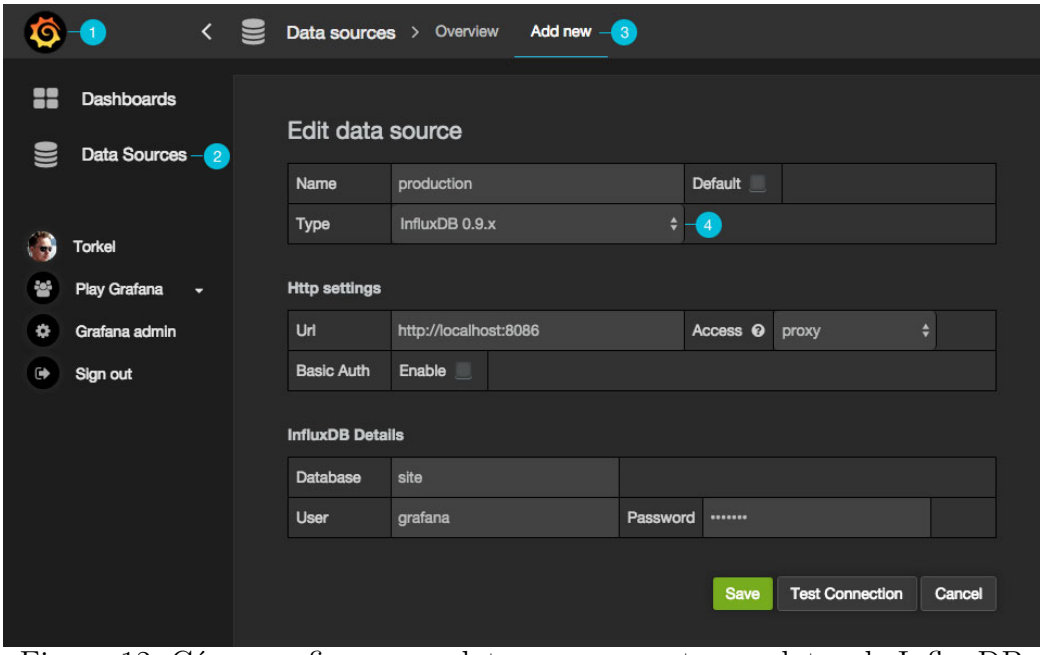

Figura 12: Cómo configurar un datasource para tomar datos de InfluxDB

- KairosDB
- OpenTSDB
- Prometheus

Para conectar Grafana con la instancia de InfluxDB configurada en el capítulo anterior, se debe comenzar por crear un datasource que consuma información de la base de datos.

En la Figura 12 se pueden ver los pasos para crear un datasource en el cliente *web* de Grafana. A continuación la explicación:

1. Abrir el menú haciendo click en el ícono de Grafana en el navegador ubicado en la parte superior de la página.
2. Una vez abierto el menú, hacer click en el enlace `DataSources`.
3. Hacer click en el botón `Add new link` en el navegador.
4. Llenar el formulario con los datos:
    - **Name:** El nombre del datasource.

- **Default:** Si este campo es activado, el datasource será preseleccionado para nuevos paneles.
- **Url:** El protocolo HTTP, IP y puerto de la API de InfluxDB
- **Access:** Si se selecciona `Proxy` , se accede a través del *backend* de Grafana. Si se selecciona `Direct`, se accede directamente desde el navegador.
- **Database:** Nombre de la base de datos de InfluxDB
- **User:** Nombre del usuario de la base de datos.
- **Password:** Contraseña del usuario de la base de datos.

Una vez creado el datasource se puede comenzar a crear tableros en Grafana.

Por ejemplo, a partir de los datos almacenados en InfluxDB, es posible crear gráficos con la intención de que sean útiles a un responsable de IT de la aplicación.

Si se quisiera medir el tiempo de respuesta de la aplicación Rails, se podría optar por un gráfico de líneas donde el eje horizontal sea la hora, y el eje vertical sea el tiempo de respuesta. Dicho gráfico podría desglosarse en tiempo de *renderizado* de las vistas, tiempo de acceso a las bases de datos, y otros tiempos.

Para poder recrear el gráfico, se podría aprovechar la información enviada desde la aplicación Rails en la sección 4.1, y realizar varias consultas a la base de datos InfluxDB.

También se podría crear un tablero donde se visualice la información generada por cAdvisor que tenga gráficos que muestren el uso de cpu, memoria, red y sistema de archivos por cada contenedor.

Usando la herramienta visual provista por Grafana es sencillo crear un tablero con la información mencionada. En la Figura 13 se puede visualizar dicho tablero.

En este capítulo se ha mostrado cómo se pueden configurar algunas herramientas de visualización para reflejar los datos que se han almacenado hasta ahora en InfluxDB y en ElasticSearch. En el capítulo siguiente se mostrará cómo crear alertas que permitan a los usuarios descubrir de forma rápida eventos a los que deberían prestar atención.

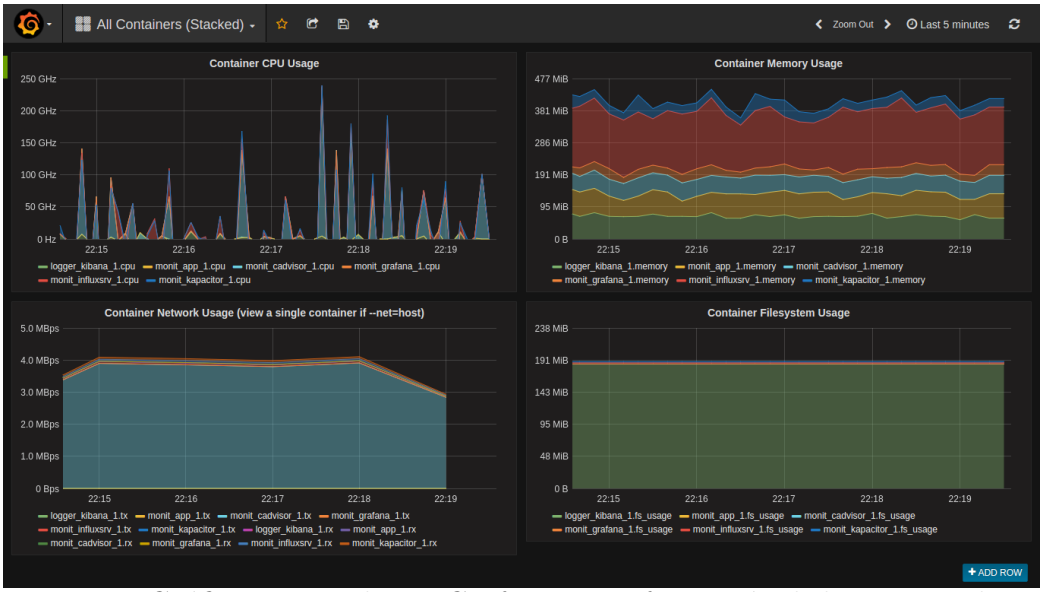

Figura 13: Gráficos generados en Grafana con información de los contenedores

6. Alertas

Cómo se ha mencionado en la sección 1.6, las alertas brindan la posibilidad de ser notificados de forma automática sobre eventos que parezcan importantes. Si no se cuenta con un sistema de alertas, la mejor forma de identificar un problema es visualizar el funcionamiento de las aplicaciones, de forma constante y manual.

Esto puede ser muy costoso, y es uno de los motivos por los que la elección de una herramienta de generación de alertas que permita describir las condiciones que deben complirse para ser notificados de situaciones importantes se torna fundamental.

En este capítulo se describirán las herramientas que han sido elegidas por ser adecuadas para resolver la importante tarea de las alertas, al mismo tiempo que se darán las razones de estas elecciones. Además se mostrará cómo configurar estas herramientas para ser notificados de eventos importantes que ocurran en la infraestructura.

En la sección 6.1 se describirá la herramienta Kapacitor, que permite la definición de alertas y el envío de las mismas a distintas fuentes, y se explicará cómo configurarla para generar importantes alertas.

En la sección 6.2 se demostrará cómo utilizar el lenguaje TICKscript para definir alertas. Primero con un ejemplo sencillo, y luego con uno más

complejo.

En la sección 6.3 se explicarán los problemas encontrados en la generación de múltiples alertas y cómo se han solucionado utilizando la herramienta de nombre Alerta.

6.1. Elección de la herramienta y funcionamiento

La herramienta Kapacitor ha sido elegida para implementar las alertas en la solución. Las razón detrás de esta decisión es que Kapacitor cuenta con una *Domain-Specific Language* (DSL) realmente simple, la cual ha permitido implementar algunas alertas complejas de forma verdaderamente rápida. Además, el hecho de que haya sido desarrollado por la misma empresa que creó InfluxDB le ha otorgado una integración excelente con el mismo.

Kapacitor es un *framework* de procesamiento de datos de código abierto, que permite crear alertas, correr procesos de extracción, transformación y carga, y detectar anomalías. Kapacitor es parte del *stack* TICK, junto con Telegraf, InfluxDB y Chronograf.

Kapacitor permite procesar datos por *streaming* y por lotes. También consultar los datos de InfluxDB en un programa y recibir datos a través del protocolo de línea y cualquier otro método que InfluxDB admita.

Adicionalmente, permite realizar cualquier transformación actualmente posible en el lenguaje de consulta InfluxQL y almacenar los datos transformados en InfluxDB. Además permite añadir funciones personalizadas definidas por el usuario para detectar anomalías.

Kapacitor puede ser configurado para notificar alertas a distintos destinos. Por ejemplo se pueden enviar a un archivo de *logs*, una URL con método HTTP POST, a través de correo electrónico y servicios como HipChat, Alerta, Sensu, Slack y PagerDuty, entre otros. Incluso es posible ejecutar un comando dirigiendo los datos de la alerta por entrada estándar (STDIN)[24].

Kapacitor cuenta con una DSL llamada TICKscript para generar alertas y definir canales de procesamiento de datos, o *pipelines*.

Un *pipeline* es un conjunto de nodos que procesa datos y aristas que conectan nodos. Los *pipelines* en Kapacitor son grafos acíclicos dirigidos, lo que significa que cada arista tiene una dirección en la que fluyen los datos y

[24] `https://docs.influxdata.com/kapacitor/v1.2/`

no puede haber ningún ciclo.

Para correr el servicio de Kapacitor simplemente es necesario ejecutar el siguiente comando en la terminal:

```
sudo service kapacitor start
```

Además, es posible usarlo como contenedor de Docker, a través de la imagen oficial de Kapacitor [25], con el comando:

```
docker pull kapacitor
```

6.2. Definición de una alerta

Una vez que se tiene instalado y configurado Kapacitor, ya es posible definir alertas.

Para esto, es necesario crear un archivo utilizando el lenguage TICKscript. A modo de ejemplo, se creará un archivo de nombre `cpu_alert.tick` con la siguiente configuración:

```
stream
    |from()
        .measurement('cpu')
    |alert()
        .crit(lambda: "usage_idle" < 70)
        .log('/tmp/alerts.log')
```

Una vez creado el archivo de configuración, es posible usarlo para definir una alerta de tipo stream con el siguiente comando:

```
kapacitor define cpu_alert -type stream -tick cpu_alert.tick -dbrp
    cadvisor.autogen
```

Con la configuración anterior cargada, Kapacitor ya tiene la información necesaria para lanzar la alerta. Para que Kapacitor comience a notificar la alerta, se la debe habilitar. Esta tarea puede realizarse con el siguiente comando:

[25] `https://hub.docker.com/_/kapacitor/`

```
kapacitor enable cpu_alert
```

Finalmente, para consultar si la alerta se encuentra cargada y visualizar los datos que estén siendo procesados, se puede ejecutar el siguiente comando:

```
kapacitor show cpu_alert
```

En este caso, este comando retornará un mensaje similar al siguiente:

```
ID: cpu_alert
ID: cpu_alert
Error:
Template:
Type: stream
Status: enabled
Executing: true
Created: 14 Mar 17 20:56 UTC
Modified: 14 Mar 17 21:22 UTC
LastEnabled: 14 Mar 17 21:21 UTC
Databases Retention Policies: ["cadvisor"."autogen"]
TICKscript:
stream
    |from()
        .measurement('cpu')
    |alert()
        .crit(lambda: "usage_idle" < 70)
        .log('/tmp/alerts.log')

DOT:
digraph cpu_alert {
graph [throughput="0.00 points/s"];

stream0 [avg_exec_time_ns="0" ];
stream0 -> from1 [processed="0"];

from1 [avg_exec_time_ns="0" ];
from1 -> alert2 [processed="0"];

alert2 [alerts_triggered="0" avg_exec_time_ns="0" crits_triggered
    ="0" infos_triggered="0" oks_triggered="0" warns_triggered="0"
    ];
}
```

Para definir alertas útiles es importante analizar las situaciones ante las cuáles podría ser importante ser avisado, y también cuál será el medio de

comunicación a través del cual llegarán las mismas.

A continuación se mostrará un ejemplo sobre cómo puede ser usado TICKscript para definir alertas más complejas.

```
stream
    |from()
        .measurement('cpu')
    |eval(lambda: 100.0 - "usage_idle")
        .as('used')
    |groupBy('service', 'datacenter')
    |window()
        .period(1m)
        .every(1m)
    |percentile('used', 95.0)
    |eval(lambda: sigma("percentile"))
        .as('sigma')
        .keep('percentile', 'sigma')
    |alert()
        .id('{{ .Name }}/{{ index .Tags "service" }}/{{ index .Tags
            "datacenter"}}')
        .message('{{ .ID }} is {{ .Level }} cpu-95th:{{ index .
            Fields "percentile" }}')
        .warn(lambda: "sigma" > 2.5)
        .crit(lambda: "sigma" > 3.0)
        .log('/tmp/alerts.log')

        .post('https://alerthandler.example.com')

        .exec('/bin/custom_alert_handler.sh')

        .slack()
        .channel('#alerts')

        .pagerDuty()

        .victorOps()
        .routingKey('team_rocket')
```

El *script* toma las medidas del porcentaje de uso de CPU, las invierte para obtener el porcentaje de tiempo en que la CPU se encuentra ociosa, las agrupa por servicio y centro de datos en ventanas de 1 minuto. Luego calcula el percentil 95 y genera alertas en caso de que se pasen determinados umbrales.

Finalmente los datos son guardados en un *log*, son enviados a un *endpoint* determinado, se ejecuta un script que maneje las alertas y se envían las alertas a Slack, a PagerDuty y a VictorOps.

Kapacitor además, permite enviar datos a InfluxDB de la siguiente manera:

```
stream
       |eval(lambda: "errors" / "total")
           .as('error_percent')
       |influxDBOut()
           .database('mydb')
           .retentionPolicy('myrp')
           .measurement('errors')
           .tag('kapacitor', 'true')
           .tag('version', '0.2')
```

Esto puede resultar muy útil en caso de que se quiera alimentar a la instancia de InfluxDB con datos procesados por Kapacitor. Incluso, se podría almacenar información de las alertas en InfluxDB para tener un seguimiento de las mismas (ver sección 8.1).

En esta sección se ha mostrado cómo usar TICKscript para definir alertas para que sean lanzadas por Kapacitor. A continuación se explicarán algunos problemas que suelen estar presentes en los sistemas de alertas y una forma en que se los puede solucionar.

6.3. Manejador de alertas

Ahora que se ha configurado Kapacitor, se cuenta con una infraestructura de monitoreo que permite generar alertas ante anomalías identificadas por condiciones previamente definidas.

Pero lanzar demasiadas alertas puede provocar una serie de situaciones no deseadas.

Una de las formas más comunes de enviar notificaciones es a través del uso de correo electrónico. Pero si se genera una alerta cuando falla la conexión a un servicio de la aplicación, y el mismo está caído, la alerta se generará todas las veces que se realice el llamado.

Si la frecuencia con la que se hace el llamado es alta, es posible que se generen un gran número de alertas. Sin embargo, para ser notificado, con una sola alerta que lea el usuario, ya debería ser suficiente.

Esto no significa que no sea deseable que la alerta se genere, ya que la cantidad de veces que la invocación al servicio falló también podría ser un dato que se querría obtener.

Es en situaciones como ésta que el envío por correo electrónico de las notificaciones generadas por las alertas puede no ser una buena decisión.

El sistema de monitoreo Alerta es una herramienta usada para consolidar alertas de múltiples fuentes y evitar su duplicación, para una visualización rápida. Con sólo un sistema es posible monitorear alertas que provengan de otras herramientas de monitoreo en una única pantalla.

Se coincidió que la herramienta Alerta es la más indicada para resolver el problema:

Alerta es una plataforma que combina un servidor de una API JSON para recibir, procesar y renderizar alertas con una simple y efectiva interfaz de usuario *web* y una herramienta de líneas de comandos. Existen numerosas integraciones con otras herramientas populares de monitoreo y es fácil agregar una propia usando directamente la API.

El acceso a la API y la herramienta de línea de comandos puede ser restringida usando claves de API y la consola usando Basic Auth o proveedores de OAuth2 como Google, GitHub y Gitlab.

Alerta acepta alertas de fuentes estándar como lo son Syslog, SNMP, Nagios, Zabbix y Sensu. Cualquier herramienta de monitoreo que pueda disparar una petición HTTP puede ser integrada de forma sencilla. Además, se pueden mandar alertas a través de *scripts* que usen la herramienta de línea de comandos.

Con Alerta es posible usar la cantidad de etiquetas que se prefieran para medir la importancia de las alertas. Para herramientas simples pueden usarse los niveles de severidad *critical*, *warning* y *ok*. Para una aplicación pueden usarse además *info* y *debug*.

Kapacitor cuenta con soporte nativo para redirigir alertas a Alerta.

Para hacer esto se debe realizar la siguiente configuración:

```
[alerta]
    enabled = true
    url = "https://alerta.yourdomain"
    token = "9hiWoDOZ9IbmHsOTeST123ABciWTIqXQVFDo63h9"
    environment = "Production"
    origin = "Kapacitor"
```

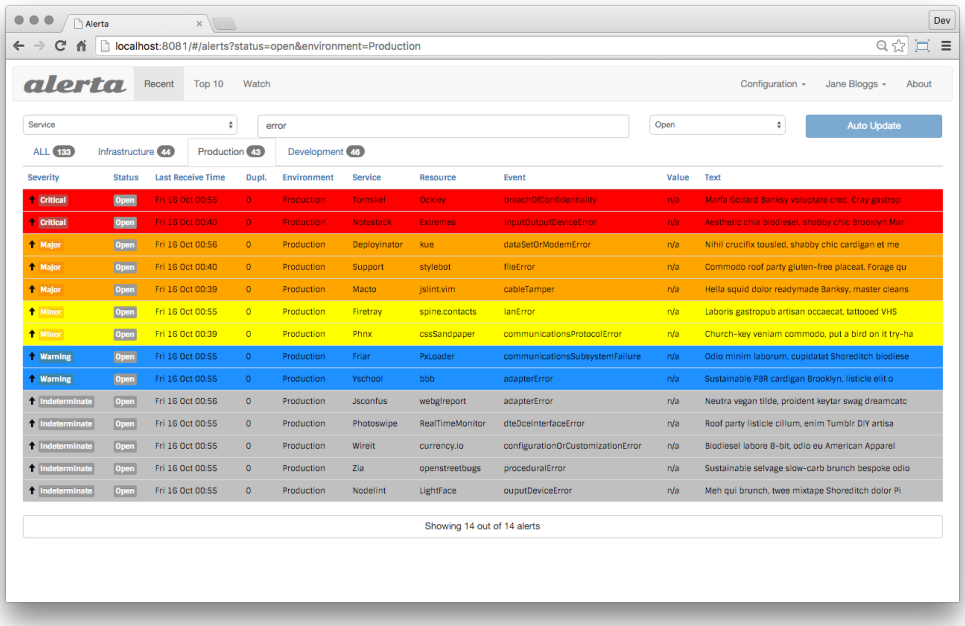

Figura 14: Cliente de Alerta agrupando alertas por prioridades

Finalmente, en TICKscript se puede enviar un mensaje a Alerta de la siguiente forma:

```
stream
    |alert()
        .alerta()
            .resource('Hostname or service')
            .event('Something went wrong')
```

Las propiedades `resource` y `event` son requeridas. Además de éstas, se puede enviar algunas propiedades opcionales:

```
stream
    |alert()
        .alerta()
            .resource('Hostname or service')
            .event('Something went wrong')
            .environment('Development')
            .group('Dev. Servers')
```

En la Figura 14 se puede ver cómo se ven las alertas listadas en el cliente de Alerta. La herramienta se encarga de ordenarlas por grado de severidad, y agruparlas, de forma que no aparezcan las alertas repetidas, sino que aparez-

can una única vez, especificando la cantidad de repeticiones con un contador.

En este capítulo se ha mostrado cómo instalar y configurar Kapacitor para generar alertas. Cómo definir alertas utilizando el lenguaje TICKscript y cómo resolver el problema de visualización y manejo de alertas agregando Alerta a la lista de herramientas.

Con ésto se ha completado la parte práctica del trabajo. A continuación se darán las conclusiones finales de la tesis.

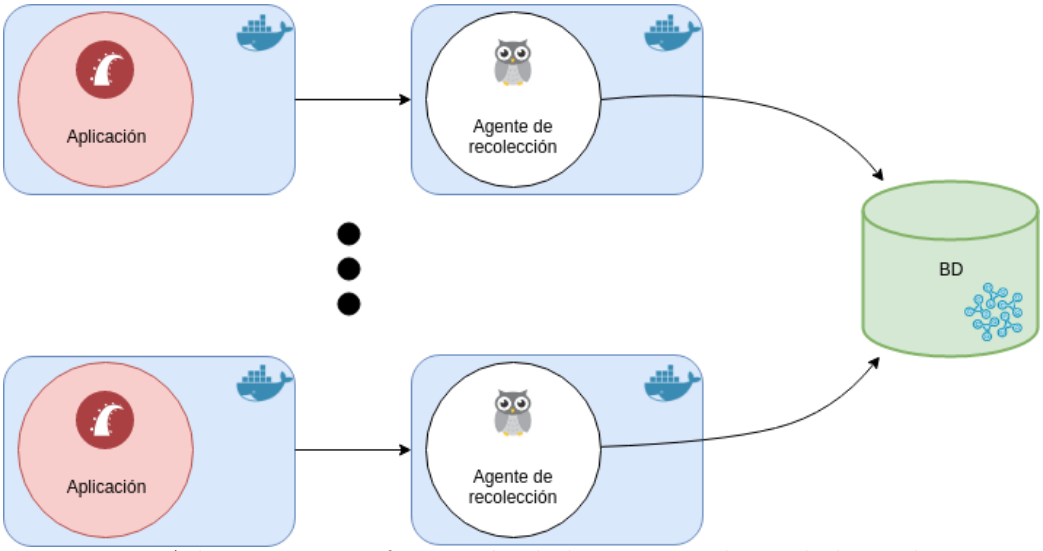

Figura 15: cAdvisor toma información de los contenedores de las aplicaciones y se la envía a InfluxDB

7. Conclusiones

A lo largo de nuestra tesis se han investigado numerosas herramientas de *software* para construir una solución de monitoreo para el CeSPI.

En primer lugar se ha configurado InfluxDB para almacenar métricas de tipo series de tiempo. Se ha mostrado cómo hacer para que las aplicaciones Rails sean ejecutadas dentro de contenedores de Docker y cómo lograr que envíen información a InfluxDB a través de la gema `influxdb-rails`.

Además se ha mostrado cómo configurar cAdvisor para obtener valiosa información del sistema y enviarla a InfluxDB. (Figura 15)

También se ha descripto cómo configurar NGINX y las aplicaciones para recuperar los datos de sus *logs* e imprimirlos en el STDOUT del contenedor de Docker para luego enviarlos a través de Fluentd a ElasticSearch. De esta forma se ha logrado utilizar toda la información guardada en *logs* que antes era desaprovechada. (Figura 16)

Se ha demostrado cómo usar Kibana para crear gráficos a partir de la información de los *logs* en ElasticSearch y realizar la exploración de los *logs* de forma sencilla, y cómo configurar un tablero de Grafana para mostrar los datos almacenados en InfluxDB.

Finalmente se ha configurado Kapacitor para generar alertas útiles, de-

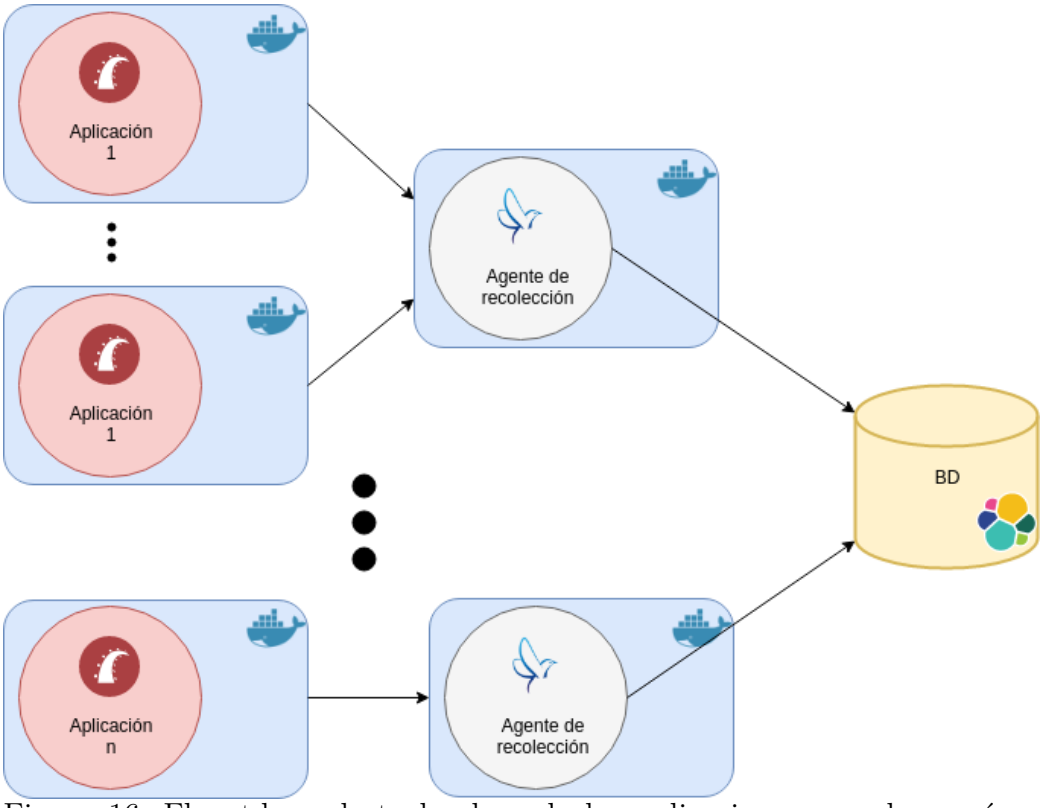

Figura 16: Fluentd recolecta los logs de las aplicaciones y se las envía a ElasticSearch

Figura 17: Kapacitor genera alertas a diferentes destinos a partir de información tomada de InfluxDB

finiendolas en el lenguaje TICKscript y luego se ha usado la herramienta Alerta para resolver el problema de múltiples alertas. (Figura 17)

Durante el transcurso de la investigación se han explorado otras herramientas como Logstash, OpenTSDB, Graphite y Telegraf, que finalmente no formaron parte de la solución.

Ninguna herramienta cumplía con todas las características que la solución propuesta demandaba, por lo que muchas veces se ha resuelto sólo tomar parte de las funcionalidades de cada herramienta para resolver el problema.

Se ha logrado construir una solución de monitoreo que se acopla al diseño de la infraestructura, y se ha hecho utilizando solamente herramientas gratuitas, de código abierto y albergadas por la misma oficina (*self-hosted*).

El sistema que se ha diseñado puede utilizarse para armar tableros y alertas que permitan mejorar el proceso de toma de decisiones para todos los roles de la organización.

Se ha construido una solución de monitoreo base para que el área de desarrollo del CeSPI pueda realizar un control básico y efectivo de sus aplicaciones, y la extienda de acuerdo a los atributos que crea necesario medir en cada aplicación.

Esta solución se ajusta perfectamente a la escalabilidad, dinamismo y automatización con los que cuenta la infraestructura.

El monitoreo es una rama de investigación que sigue creciendo cada día. Si bien actualmente existen varias herramientas relacionadas con el monitoreo, creemos que en un futuro se contará con herramientas más completas.

Durante el desarrollo de la tesis se ha tenido que investigar sobre disciplinas ajenas a la informática, como lo son la administración de proyectos, la estadística, la visualización efectiva de información e incluso la psicología.

Hemos aprendido acerca de la importancia del monitoreo para resolver problemas rápidamente, estudiar el comportamiento de los sistemas y aplicaciones y tomar decisiones para mejorar la implementación de los procesos de desarrollo, la infraestructura de *software* y las aplicaciones.

8. Trabajos futuros

Durante el desarrollo de la investigación se ha ayudado a resolver un problema más o menos específico de monitoreo para el CeSPI. Pero el mundo del monitoreo tiene un alcance bastante mucho mayor.

En este capítulo se mencionarán brevemente algunas investigaciones que podrían llevarse a cabo partiendo del contenido de este trabajo.

8.1. Monitoreo del sistema de alertas

Los sistemas de alertas pueden generar información muy importante. Tener un seguimiento de las alertas en sistemas de monitoreo propios podría permitir por ejemplo realizar consultas sobre la cantidad de alertas generadas por período de tiempo.

Tener tableros de control que muestren este tipo de información, podría ser útil para jefes de proyecto. Las alertas generadas podrían ser una buena métrica para ver la estabilidad de la aplicación. Por ejemplo se podría correlacionar la generación de alertas con la cantidad de memoria, o el procesamiento de las CPUs a lo largo del tiempo.

Podrían utilizarse algunas herramientas que se han mencionado a lo largo de la tesis para tomar información de Kapacitor y con esa información crear tableros en tiempo real que permitan mantener un seguimiento de la generación de alertas durante la evolución de las aplicaciones.

8.2. Monitoreo de despliegue de aplicaciones

Se ha mencionado en la sección 2.1 que el CeSPI sigue una metodología de trabajo fuertemente inspirada en DevOps. Un posible trabajo futuro es mantener un seguimiento del despliegue de las aplicaciones.

En empresas en donde se utiliza una metodología de despliegue continuo, podría ser muy útil tener tableros de mando que permitan ver información sobre las versiones de la aplicación a lo largo del tiempo.

Por ejemplo, se podrían marcar hitos en los tableros que diferencien una versión de la aplicación de la siguiente, y sería fácil contrastar cómo mejora o empeora el rendimiento de la aplicación luego de cada lanzamiento.

8.3. Explorar otras herramientas

La solución de monitoreo para el CeSPI a dejado de lado algunas herramientas que podrían ser de utilidad para casos similares. Se han descripto brevemente algunas de estas herramientas en el anexo B.

Sería interesante explorar cómo configurar estas herramientas y otras herramientas nuevas para agregar nuevas funcionalidades a la infraestructura actual.

8.4. Monitoreo de la infraestructura del monitoreo

El sistema de monitoreo es un sistema de *software*, y como tal, no es inmune a fallos, *bugs* o comportamientos inesperados. Puede ser importante tener un control sobre el mismo sistema de monitoreo, para asegurar que continúe en funcionamiento y que no se pierda información.

Un posible trabajo a futuro es explorar cómo configurar un sistema de monitoreo que permita visualizar el sistema de monitoreo de la infraestructura.

Este sistema debería cumplir con la condición de ser tolerante a fallos, y de poder revivir a la infraestructura de monitoreo principal en caso de que ésta dejara de funcionar por algún motivo.

8.5. Monitoreo que aprenda sobre sí mismo

Como se ha mencionado anteriormente, el monitoreo es una rama de investigación que está en constante crecimiento. El uso de algoritmos de *machine learning* en el monitoreo es uno de los aspectos que está tomando más fuerza.

Allois Reitbauer explica: "La construcción de sistemas de monitoreo de auto-aprendizaje ayuda a los equipos de operaciones a concentrarse en sus tareas centrales en lugar de intentar interpretar un tablero de gráficos. El monitoreo inteligente también está en el núcleo del movimiento DevOps" [12, Prefacio]

El uso de técnicas de auto-aprendizaje es un posible trabajo futuro. Estas técnicas pueden traer grandes beneficios a las organizaciones, ayudándolas a identificar problemas rápidamente, realizando esta tarea con mayor precisión en cada iteración e incluso adelantándose a que los problemas ocurran.

Anexos

A. Tecnologías del CeSPI

En esta sección se describirán aspectos de algunas herramientas usadas por el CeSPI, que si bien no ha sido necesario su conocimiento a fondo para entender el trabajo, han formado parte de la investigación y por eso es importante que sean mencionadas.

A.1. Ruby

Ruby es un lenguaje de programación interpretado y orientado a objetos, creado en 1995 por Yukihiro Matsumoto. Ruby combina una sintaxis inspirada en Python y Perl, con características de programación orientada a objetos similares a Smalltalk. Su implementación oficial es distribuida bajo una licencia de *software* libre.

En Ruby, todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas. Toda función es un método y las variables siempre son referencias a objetos, y no los objetos mismos.

Ruby soporta herencia con enlace dinámico, mixins y métodos definidos por instancia. Ruby no soporta herencia múltiple, pero esta se puede imitar haciendo que una clase importe módulos como si fueran mixins.

Ruby es un lenguaje multiparadigma, en el sentido en que permite programación procedural, con orientación a objetos y funcional. Todas las sentencias tienen valores, y las funciones devuelven la última evaluación. Ruby soporta introspección, reflexión y metaprogramación [19].

A.2. Ruby on Rails

Ruby on Rails, también conocido como RoR o Rails, es un framework de aplicaciones *web* de código abierto escrito en el lenguaje de programación Ruby, siguiendo el patrón Modelo Vista Controlador (MVC).

Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el

formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby [14].

A.3. Docker

Docker es un proyecto de código abierto que permite la automatización del despliegue de aplicaciones dentro de contenedores de *software* en Linux. Los contenedores proporcionan una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo.

Docker utiliza características de aislamiento de recursos del kernel de Linux, tales como cgroups y espacios de nombres para permitir que contenedores independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

Docker es una herramienta que puede empaquetar una aplicación y sus dependencias en un contenedor virtual que se puede ejecutar en cualquier servidor Linux. Esto otorga flexibilidad y portabilidad al espacio donde la aplicación se ejecuta, por ejemplo en instalaciones físicas, una nube pública o nubes privadas.

Docker implementa una interfaz para proporcionar contenedores livianos que ejecutan procesos de forma aislada.

A diferencia de una máquina virtual, un contenedor Docker no requiere incluir un sistema operativo independiente. En su lugar se basa en las funcionalidades del kernel y utiliza el aislamiento de recursos (CPU, memoria, bloque de entrada y salida y la red, entre otros) y espacios de nombres separados para aislar la aplicación del sistema operativo.

A partir del uso de contenedores, los recursos pueden ser aislados y los servicios restringidos. Se otorga a los procesos la capacidad de tener una visión casi completamente privada del sistema operativo, con su propio identificador de espacio de proceso, la estructura del sistema de archivos y las interfaces de red. Múltiples contenedores comparten el mismo núcleo, pero cada contenedor puede ser restringido a usar sólo una cantidad definida de recursos.

En la práctica, Docker puede ser utilizado para simplificar la creación de sistemas altamente distribuidos. El despliegue de nodos puede realizarse a medida que se disponga de recursos o cuando se necesiten más nodos, lo que permite la construcción de una plataforma como servicio. Docker también facilita el armado y funcionamiento de tareas de carga de trabajo [5].

A.4. Docker Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor. Con Compose, es posible definir la configuración de los servicios de una aplicación en un archivo, y luego crear e iniciar todos los servicios desde esa configuración con un único comando.

Compose ha demostrado ser una excelente herramienta para desarrollar y probar ambientes de desarrollo, y para definir flujos de integración continua [4]. Para usar Compose, se necesitan seguir los siguientes pasos:

- Definir el ambiente de una aplicación con un Dockerfile [26], para poder reproducirlo en cualquier lugar.
- Definir los servicios que constituyen la aplicación en un archivo llamado docker-compose.yml[27], de forma que puedan ser ejecutados juntos en un ambiente aislado.
- Ejecutar el comando `docker-compose up` y Compose iniciara y correrá la aplicación en su totalidad[28].

A.5. Rancher

Rancher es una plataforma de código abierto para manejar contenedores que facilita la tarea de correr cualquier aplicación basada en contenedores en cualquier infraestructura. Soporta Kubernetes, Mesos y Docker Swarn.

Fue diseñado para resolver todos los desafíos críticos necesarios para correr aplicaciones en contenedores. Rancher provee un conjunto completo de servicios de infraestructura para contenedores, incluyendo servicios de red, servicios de almacenamiento, manejo de hosts y balanceo de carga. Todos estos servicios funcionan sobre cualquier infraestructura y facilitan la tarea de desplegar y manejar aplicaciones de forma confiable [15].

Rancher consiste en cuatro componentes principales:

---

[26]Cf. `https://docs.docker.com/engine/reference/builder/`

[27]Cf. `https://docs.docker.com/compose/compose-file/`

[28]Cf. `https://docs.docker.com/compose/reference/up/`

- **Orquestación de infraestructura:** Rancher toma recursos informáticos de cualquier nube pública o privada en la forma de hosts de Linux. Cada host de Linux puede ser una máquina virtual o física. Rancher no espera más de cada host que CPU, memoria, almacenamiento en el disco local y conectividad en la red.

  Rancher implementa una capa portable de servicios de infraestructura diseñados específicamente para alimentar aplicaciones en contenedores. Los servicios de infraestructura de Rancher incluyen servicios de red, almacenamiento, balance de carga, DNS y seguridad.

  Los servicios de infraestructura de Rancher son típicamente desplegados como contenedores, y de esta forma el mismo servicio de infraestructura de Rancher puede correr en cualquier host de linux en una nube.

- **Orquestación y programación de contenedores:** Muchos usuarios eligen correr aplicaciones en contenedores usando un framework de orquestación y programación de contenedores. Rancher incluye una distribución de todos los *frameworks* populares hoy en día, incluyendo Docker Swarm, Kubernetes y Mesos. Un mismo usuario puede crear múltiples *clusters* de Swarm o Kubernetes, y luego usar las herramientas nativas de estos *frameworks* para manejar sus aplicaciones.

  Además de Swarm, Kubernetes, y Mesos, Rancher soporta su propio framework de orquestación y programación de contenedores, llamado Cattle. Cattle fue originalmente diseñado como una extensión de Docker Swarm, pero fue divergiendo a medida que continuó el desarrollo de ambos.

- **Catálogo de aplicación:** Un usuario de Rancher puede desplegar una aplicación multicontenedor en *clusters* desde el catálogo de aplicación presionando un único botón. Los usuarios pueden manejar las aplicaciones desplegadas y ejecutar actualizaciones completamente automatizadas cuando nuevas versiones de la aplicación se vuelven disponibles. Rancher mantiene un catálogo público que consiste en aplicaciones populares creadas por la comunidad de Rancher. Un usuario puede crear su propio catálogo privado.

- **Control de nivel empresarial:** Rancher soporta *plugins* de autenticación de usuarios flexibles y viene por defecto con integración de autenticación de usuarios con ActiveDirectory, LDAP y Github. Rancher soporta Control de Acceso basado en Roles (RBAC) a nivel de ambientes, permitiendo a los usuarios y grupos compartir o denegar acceso a, por ejemplo, ambientes de desarrollo y producción.

B. Otras Herramientas

Durante la investigación se han explorado varias herramientas de monitoreo que finalmente no fueron elegidas para la solución final, debido a diferentes factores, como por ejemplo el hecho de que algunas de estas son herramientas privativas, o que no se ajustaban a los problemas, o que eran *frameworks* completos de los cuales sólo servía un pequeño subconjunto de su funcionalidad.

A continuación se explicarán un poco más a fondo estas herramientas, con el propósito de dar a conocer otras herramientas que han sido parte de la investigación y que resuelven varios problemas similares a los que se han mencionado durante la tesis.

B.1. Prometheus

Prometheus es un sistema de monitoreo de servicios y sistemas creado por *Cloud Native Computing Foundation*. Prometheus recolecta métricas de objetivos configurados a intervalos dados, evalúa expresiones, muestra resultados y activa alertas si observa que alguna condición es verdadera [13].

Las prestaciones más distinguidas de Prometheus son:

- Un modelo de datos multidimensional, es decir series de tiempo definidas por un nombre de métrica y un conjunto de dimensiones clave-valor.
- Un lenguaje de consultas flexible.
- No depende de un almacenamiento distribuido.
- Almacenamiento de series de tiempo en memoria y en disco local en un formato eficiente.
- La recolección de datos de series de tiempo ocurre a través de un modelo pull por sobre HTTP.
- Los objetivos son encontrados a través de configuración estática o descubrimiento de servicios.
- Múltiples modos de representación gráfica, soporte de tableros e integración con Grafana

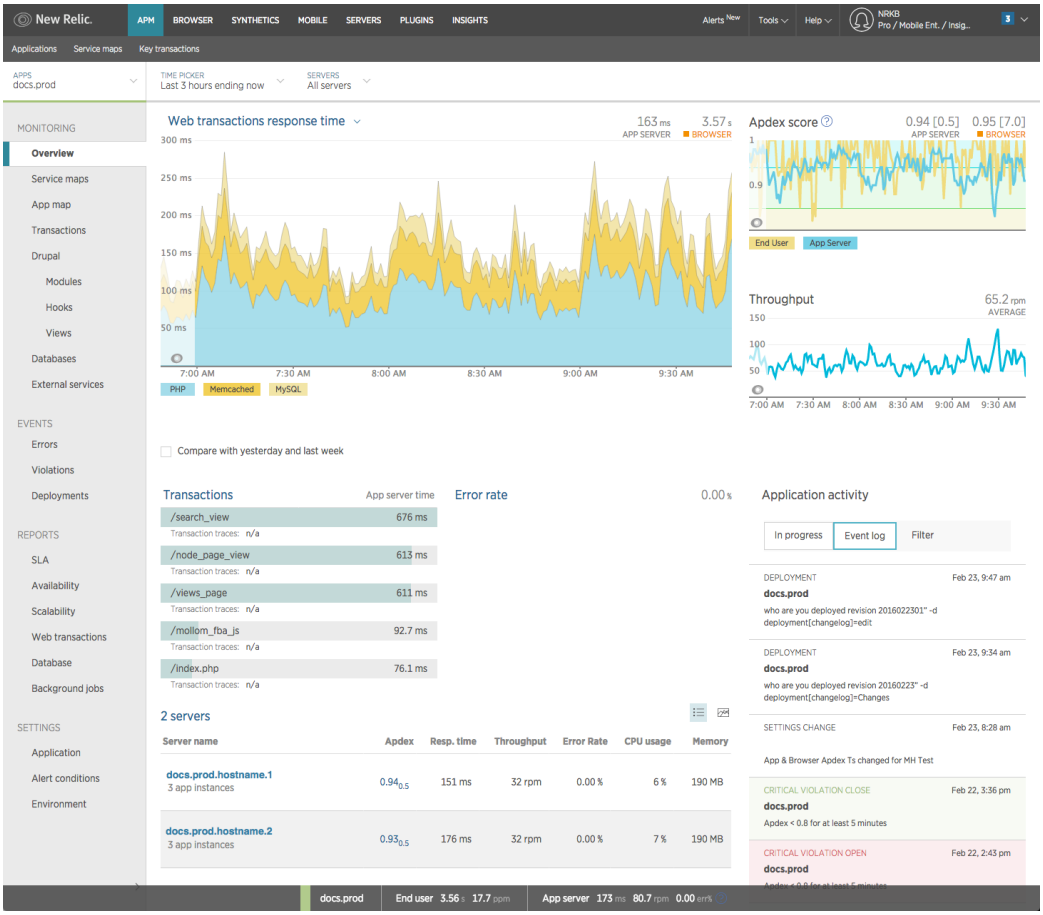

Figura 18: Cliente web de New Relic

- Gran conjunto de librerías para los clientes en varios lenguajes, que permite una sencilla instrumentación de servicios.
- Integración con Docker, HAProxy, StatsD, entre otros.

B.2. New Relic APM

El producto de *software* de monitoreo de rendimiento de aplicaciones de New Relic envía datos en tiempo real sobre el rendimiento de las aplicaciones *web* y el nivel de satisfacción que experimentan los usuarios.

Con seguimiento de transacciones de punto a punto y una variedad de reportes y gráficos a color, APM visualiza los datos hasta los niveles más profundos. Con APM es posible identificar de forma rápida problemas potenciales antes que estos afecten a los usuarios finales. Figura 18

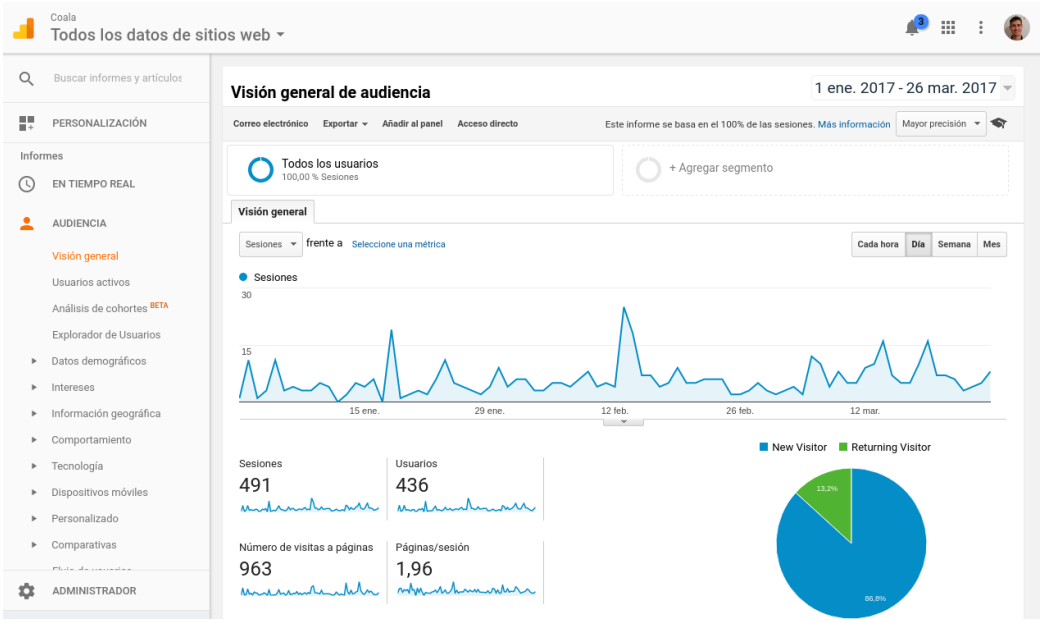

Figura 19: Cliente web de Google Analytics

APM es *software* privativo y cuenta con varios planes pagos, según los requerimientos de los clientes. Dependiendo del nivel de subscripción, es posible tomar ventaja de diversas prestaciones.

New Relic tiene una excelente integración con el lenguaje de programación Ruby [16].

B.3. Google Analytics

Google Analytics es una herramienta de analítica *web* que ofrece información agrupada del tráfico que reciben los sitios *web*, según factores como características de la audiencia, el comportamiento de los usuarios y las conversiones o hitos alcanzados en el sitio *web*.

Fue creada por Google a partir de un producto anterior comprado por la empresa, llamado Urchin.

Con Analytics se pueden obtener informes sobre el rendimiento de las páginas, los resultados de campañas de marketing online, cantidad de sesiones agrupadas por fuente del tráfico, tasas de rebote, duración de las sesiones y contenidos visitados, entre otras. Figura 19

Para usar esta herramienta en el desarrollo *web*, basta con añadir un código JavaScript en cada una de las páginas *web* que se desea analizar.

Google Analytics cuenta con una interfaz muy completa de informes con gráficos [6].

B.4. Piwik

Piwik es un programa completo de PHP y MySQL que puede ser descargado e instalado por los programadores en su servidor *web*. El proceso de instalación no toma más de 5 minutos. Luego de instalar Piwik se obtiene un código de Javascript, que se puede copiar y pegar en sitios *web* de los que se desea hacer un seguimiento y acceder a sus reportes de análisis en tiempo real.

Piwik dice ser una alternativa de *software* libre a Google Analytics, y al momento de escribir esta tesis es usado por más de 1.000.000 de sitios *web*.

Piwik provee reportes de analiticas *web* en tiempo real. Para sitios *web* de mucho tráfico, es posible elegir la frecuencia con la que los reportes deben ser procesados.

Como Piwik se encuentra instalado en un servidor personal, los datos son almacenados en una base de datos propia. Esto significa que uno es dueño de sus propios datos.

Piwik es *software* libre, y puede ser fácilmente configurado para respetar la privacidad de los visitantes del sitio *web*. Piwik tiene una comunidad de más de 200.000 usuarios activos.

Piwik es moderno, y tiene una interfaz que lo hace fácil de utilizar. Es posible personalizar los tableros y los widgets. Figura 20

Piwik tiene avanzadas capacidades de análisis *web*, como seguimiento de *e-commerce*, seguimiento de metas, seguimiento de campaña, variables personalizadas, reportes de emails, localización espacial y mapas en tiempo real [11].

B.5. Datadog

Datadog es un servicio de monitoreo para aplicaciones en la nube, que junta datos de servidores, bases de datos, herramientas y servicios, y los presenta en una vista unificada de un *stack* entero. Estas capacidades son provistas en una plataforma de análisis de datos basada en SaaS (Software como un servicio).

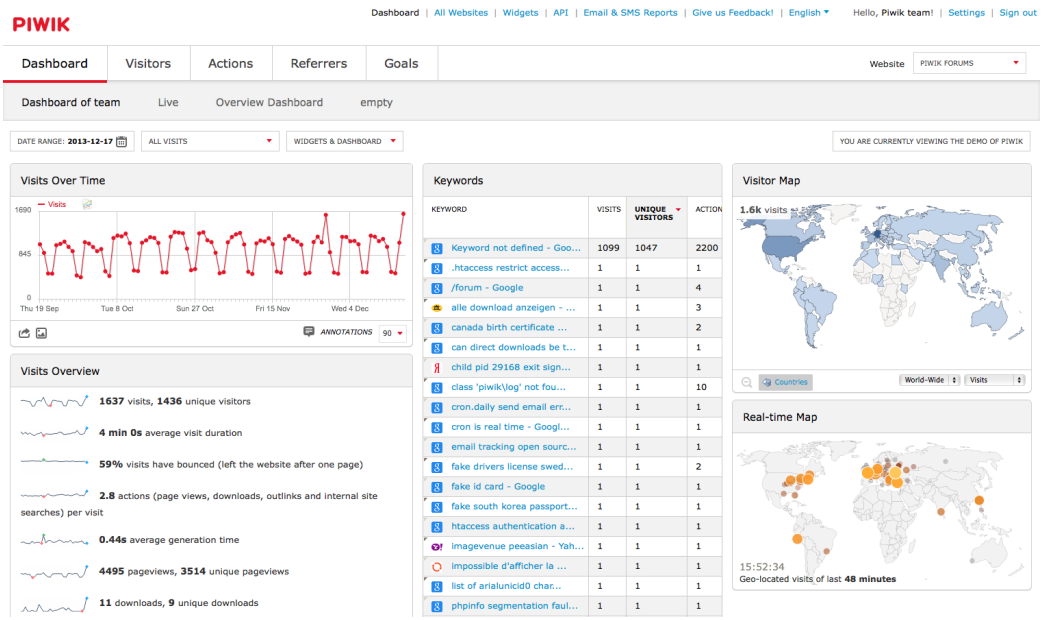

Figura 20: Cliente web de Piwik

Datadog usa un agente de código abierto escrito en Python para recolectar métricas y eventos. Su backend está construido usando un número de tecnologías de código abierto y cerrado, incluyendo D3, Apache Cassandra, Kafka y PostgreSQL.

Datadog ayuda a los desarrolladores y equipos de IT a ver la infraestructura en un sólo lugar, incluyendo la nube, los servidores, las aplicaciones, los servicios, las métricas y más.

Datadog incluye tableros interactivos en tiempo real que pueden ser personalizados para las necesidades específicas de un equipo de trabajo. También cuenta con la capacidad de realizar búsquedas de texto completo para métricas y eventos, herramientas de colaboración de equipos de trabajo, alertas para problemas críticos y una API de acceso.

Datadog integra varias herramientas de *software*, de forma que los flujos de trabajo de equipos no sean modificados ni interrumpidos al adoptar el servicio de Datadog [2].

B.6. Riemann

Riemann permite la agregación de eventos provenientes de servidores y aplicaciones utilizando un poderoso lenguaje de procesamiento de *streaming*.

Puede enviar un email por cada excepción ocurrida en las aplicaciones, mantener un seguimiento de la distribución de latencia en las aplicaciones *web*, y observar los procesos principales de cualquier host, por memoria y CPU.

Puede combinar estadísticas de todos los nodos Riak en un *cluster*, reenviar la información a Graphite, y mantener un seguimiento de la actividad de los usuarios, segundo a segundo.

Los *streams* de Riemann son simplemente funciones que aceptan un evento. Los eventos son estructuras con algunos campos comunes, como host y service. Es posible usar docenas de *streams* integrados en Riemann para filtros, alertas y combinación de eventos, y también escribir los propios.

La configuración de Riemann es un programa de Clojure, y por esto su sintaxis es concisa, regular y extensible. La configuración como código minimiza repeticiones en código, y otorga la flexibilidad para adaptarse a situaciones complejas [17].

C. Ejemplo de docker-compose.yml

Para realizar la obtención de datos del sistema, contenedores o aplicaciones explicadas en capítulo 3 se ha utilizado el siguiente archivo `docker-compose.yml` que levanta todas las herramientas propuestas:

```
version: "2"
services:

  app:
    build: app
    command: rails server -p 3000 -b '0.0.0.0'
    volumes:
      - ./app/:/app
    ports:
      - "8080:3000"
    networks:
      - lognet

  influxsrv:
    image: influxdb:1.0.0-rc1
    ports:
      - "8083:8083"
      - "8086:8086"
    expose:
      - "8090"
      - "8099"
    environment:
      - PRE_CREATE_DB=cadvisor
    networks:
      - metricsnet

  cadvisor:
    image: google/cadvisor:v0.24.0
    command: -storage_driver=influxdb -storage_driver_db=cadvisor -
       storage_driver_host=influxsrv:8086
    ports:
      - "9090:8080"
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys:ro
```

```
      - /var/lib/docker/:/var/lib/docker:ro
    networks:
      - metricsnet

  grafana:
    image: grafana/grafana:3.1.1
    ports:
      - "3000:3000"
    environment:
      - INFLUXDB_HOST=localhost
      - INFLUXDB_PORT=8086
      - INFLUXDB_NAME=cadvisor
      - INFLUXDB_USER=root
      - INFLUXDB_PASS=root
    networks:
      - metricsnet

networks:
  metricsnet:
    driver: bridge
```

Se copia a continuación el contenido del archivo `docker-compose.yml` completo de ejemplo para la configuración de una aplicación Rails con FluentD, NGINX, ElasticSearch y Kibana.

```
version: "2"

services:

  fluentd:
    image: fluent/fluentd:latest
    ports:
      - "24224:24224"
    volumes:
      - ./fluentd/etc:/fluentd/etc
    command: /fluentd/etc/start.sh
    networks:
      - lognet

  elasticsearch:
    image: elasticsearch
    ports:
      - "9200:9200"
```

```
    - "9300:9300"
  volumes:
    - ./elasticsearch/data:/usr/share/elasticsearch/data
  environment:
    ES_JAVA_OPTS: "-Xms2g -Xmx2g"
  networks:
    - lognet

kibana:
  image: kibana
  restart: always
  ports:
    - "5601:5601"
  environment:
    - ELASTICSEARCH_URL=http://elasticsearch:9200
  networks:
    - lognet

nginx:
  image: nginx
  ports:
    - "80:80"
  logging:
    driver: fluentd
    options:
      tag: "nginx.docker.{{.Name}}"
  networks:
    - lognet

app:
  build: app
  command: rails server -p 3000 -b '0.0.0.0'
  volumes:
    - ./app/:/app
  ports:
    - "3000:3000"
  logging:
    driver: fluentd
    options:
      tag: "rails.docker.{{.Name}}"
  networks:
    - lognet
```

```
networks:
  lognet:
    driver: bridge
```

Referencias

[1] Kevin J. Schmidt Anton A. Chuvakin. *Logging and log management: The authoritative guide to understanding the concepts surrounding logging and log management*. Syngress, 1 edition, 2012.

[2] Datadog. Infrastructure and application monitoring as a service. `https://www.datadoghq.com/product/`. Visitado en 20/03/2017.

[3] Jason Dixon. *Monitoring with Graphite*. O'Reilly Media, 2015.

[4] Docker. Overview of docker compose - docker documentation. `https://docs.docker.com/compose/`. Visitado en 08/11/2016.

[5] Docker. What is docker? `https://www.docker.com/what-docker`. Visitado en 05/11/2016.

[6] Google. Sitio web oficial de google analytics: Analítica web e informes. `https://www.google.com.ar/intl/es_ALL/analytics/index.html`. Visitado en 14/03/2017.

[7] InfluxDB. Scalable datastore for metrics, events, and real-time analytics. `https://github.com/influxdata/influxdb`. Visitado en 10/12/2016.

[8] Murugiah Souppaya Karen Kent. Guide to computer security log management. Technical report, National Institute of Standards and Technology, 2006.

[9] Slawek Ligus. *Effective Monitoring and Alerting*. O'Reilly Media, 2012.

[10] OpenTSDB. A distributed, scalable monitoring system. `http://opentsdb.net/overview.html`. Visitado en 20/03/2017.

[11] Piwik. What is piwik? - analytics platform. `https://piwik.org/what-is-piwik/`. Visitado en 15/03/2017.

[12] Baron Schwartz Preetam Jinka. *Anomaly Detection for Monitoring*. O'Reilly Media, 1 edition, 2015.

[13] Prometheus. Overview | prometheus. `https://prometheus.io/docs/introduction/overview/`. Visitado en 04/11/2016.

[14] Ruby On Rails. A web-application framework that includes everything needed to create database-backed web applications according to the model-view-controller (mvc) pattern. `http://rubyonrails.org/`. Visitado 05/01/2017.

[15] Rancher. Build your own private container service. `http://rancher.com/`. Visitado en 10/11/2016.

[16] New Relic. New relic apm | new relic documentation. `https://docs.newrelic.com/docs/apm/new-relic-apm`. Visitado en 21/02/2017.

[17] Riemann. Core concepts. `http://riemann.io/concepts.html`. Visitado en 01/12/2016.

[18] Eric Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* Crown Business, 2011.

[19] Ruby. Acerca de ruby. `https://www.ruby-lang.org/es/about/`. Visitado en 05/01/2017.

[20] James Turnbull. *The Art of Monitoring.* James Turnbull, pdf edition, 2016.

Glosario

Alerta

sistema de administración de alertas. 73, 78–80, 83

API

Application Programming Interface. 24, 32–38, 40, 41, 44, 48, 49, 71, 78, 95

BigQuery

servicio *web* que permite análisis interactivo de grandes conjuntos de datos. 42

BSD

Berkeley Software Distribution. 47

cAdvisor

herramienta para analizar recursos y rendimiento de contenedores en ejecución. 32, 40–43, 71, 81

Capistrano

herramienta para automatización y despliegue escrita en Ruby. 27, 28

carbon

servicio que recepta datos de series de tiempo. 33

CeSPI

Centro Superior para el Procesamiento de la Información. 1–3, 24–26, 29, 37, 40, 44, 46, 51, 81, 83–85, 87

Chef

plataforma que automatiza la configuración de la infraestructura. 27

contenedor

tecnología de virtualización en el nivel de sistema operativo. 28, 30, 32, 38–41, 43–45, 47, 48, 56, 57, 65, 71, 74, 81, 88–90, 97

CPU

Central Processing Unit. 12, 21, 29, 76, 96

datasource

Fuente de donde puede tomar datos Grafana para realizar consultas. 62, 69–71

CD

Continuous delivery. Buscar un buen enlace. 25

desviación estándar

es una medida de dispersión para variables de razón y de intervalo. Se define como la raíz cuadrada de la varianza de la variable. 7

DevOps

develpment-operations. Se trata de una cultura o movimiento que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales en las tecnologías de la información (IT). 1, 2, 25, 84, 85

Docker

plataforma de contenedores de software. 28, 30, 32, 38–41, 43–45, 47, 48, 51, 56, 64, 74, 81, 88

Dockerfile

archivo con instrucciones para construir una imágen de Docker de forma automática. 38, 40

DockerHub

sitio con repositorios oficiales de Docker. 48

DSL

Domain-Specific Language. 73

ElasticSearch

base de datos distribuida orientada a documentos optimizada para búsquedas full-text. Ver más: `https://www.elastic.co/products/elasticsearch`. 27, 42, 44, 48–52, 56, 57, 62–64, 66, 68, 69, 71, 81, 82, 98

estadístico

medida cuantitativa, derivada de un conjunto de datos de una muestra, con el objetivo de estimar o inferir características de una población o modelo estadístico. 6–8

Fluentd

colector de datos de código abierto, que le permite unificar la recopilación y el consumo de datos. Ver más: `http://www.fluentd.org/architecture`. 44, 48, 50–52, 54, 56, 57, 61, 81, 82

gema

las gemas son plugins y/o códigos añadidos a proyectos Ruby on Rails, que permiten extenderlos con nuevas funcionalidades de manera modular. 29, 35, 36, 38, 39, 45, 46, 51, 52, 81

Go

lenguaje de programación concurrente y compilado inspirado en la sintaxis de C. 32, 41, 51

Grafana

herramienta de monitoreo para visualización de datos. 62–64, 69–72, 81

Graphite

herramienta *open source* para monitorear y gráficar el rendimiento de sistemas de computación. 32, 33, 69, 83

graphite-web

cliente web y API de Graphite. 33

HBase

base de datos distribuida, no relacional, de código abierto. 34

Heka

herramienta para la recolección y procesamiento de datos. 50, 51

histograma

representación gráfica de una distribución de frecuencias por medio de rectángulos, cuyas anchuras representan intervalos de la clasificación y cuyas alturas representan las correspondientes frecuencia. `http://dle.rae.es/?id=KWdW8dL`. 7, 8, 66

host

computadora conectada a una red, que provee y utiliza servicios de la misma. 3, 13, 15, 34, 36, 40, 90, 96

htop

programa para visualizar procesos de forma interactiva. `http://hisham.hm/htop/` sort. 29

HTTP

Hypertext Transfer Protocol. 11, 16, 22, 29, 32, 35, 47, 49, 55, 61, 71, 73, 78

InfluxDB

base de datos de tipo series de tiempo de código abierto. 32–40, 42, 62, 63, 69–71, 73, 77, 81, 82

IP

Internet Protocol. 12, 13, 37, 71

IT

Information Technology. 12, 13, 20, 37, 71, 95

sistema de seguimiento de incidentes

paquete de software que administra y mantiene listas de incidentes, conforme son requeridos por una institución. 21

JSON

JavaScript Object Notation. 15, 33, 48, 49, 51, 54, 55, 78

Kafka

plataforma de código abierto para el procesamiento de *stream* de datos. 42, 95

Kapacitor

framework de procesamiento de datos que permite generar alertas. 72–74, 77, 78, 80–82, 84

Kibana

herramienta de visualización para Elasticsearch. 49, 62–66, 68, 69, 81

Lean

Buscar un buen enlace. 25, 26

Linux

término empleado para referirse a la combinación del sistema operativo GNU, y el núcleo (*kernel*) Linux. 21, 28, 69, 90

lograge

gema de Rails para volver sus logs más útiles. 45, 46

Logstash

recolecta, analiza y transforma logs. Ver más: `https://www.elastic.co/products/logstash`. 49–51, 83

MongoDB

base de datos de código abierto orientada a documentos. 27

MySQL

sistema de gestión de bases de datos relacional. `https://www.mysql.com/` sort. 27, 29, 94

Nagios

sistema de monitorización ampliamente utilizado. 30

NGINX

proxy reverso http, mail server y proxy reverso generico TCP/UDP. Ver más: `http://nginx.org/en/`. 44, 47, 48, 55, 57, 61, 81

OpenTSDB

base de datos de series de tiempo escalable y distribuida. 32, 34, 70, 83

percentil

medida de posición usada en estadística que indica, una vez ordenados los datos de menor a mayor, el valor de la variable por debajo del cual se encuentra un porcentaje dado de observaciones en un grupo de observaciones. 7, 8, 22, 67, 76

PHP

Hypertext Preprocessor `http://php.net/manual/es/intro-whatis.php`. 24, 94

ping

utilidad diagnóstica en redes de computadoras que comprueba el estado de la comunicación del host local con uno o varios equipos remotos de una red IP. 11, 21

Prometheus

sistema de monitoreo y bases de datos de series de tiempo. 42, 70, 91

Rancher

plataforma para manejar y desplegar contenedores en producción. 28

Redis

motor de base de datos en memoria. 27, 42

Rails

framework Ruby enfocado al desarrollo de aplicaciones web. `https://rubyonrails.org/` first. 27, 29, 32, 35–40, 44–48, 57, 60, 68, 71, 81, 87, 98

RPC

Remote Procedure Call. 35

Ruby

`https://www.ruby-lang.org/es/`. 24, 27, 36, 49–51, 87

SNMP

Simple Network Management Protocol. 15, 16, 78

Splunk

herramienta de monitoreo y análisis de datos. 50, 51

SQL

Structured Query Language. 37

SSH

Secure Shell. Programa que permite acceder a máquinas remotas a través de una red first. 29

stake trace

pila de llamadas. 30

StatsD

demonio para la agregación de estadísticas. 42

STDERR

error estándar. 48

STDIN

entrada estándar. 73

STDOUT

salida estándar. 42, 44, 48, 81

swap

zona del disco (un fichero o partición) que se usa para guardar las imágenes de los procesos que no han de mantenerse en memoria física. 12, 29

Syslog

protocolo de red y aplicaición utilizados para el manejo de logs. 13, 15

TCP

Transmission Control Protocol. 35, 54

Telegraf

agente de recolección y procesamiento de métricas. 40, 41, 73, 83

TICKscript

DSL utilizada para definir los pipelines y el procesamiento de datos en Kapacitor. 72–74, 76–78, 80, 83

UNLP

Universidad Nacional de La Plata. 1, 24, 36, 37

URL

Uniform Resource Locator. 55, 73

whisper

base de datos para series de tiempo. 33

XML

Extensible Markup Language. 15, 33

Índice de figuras

Listado de bloques de código

Índice de cuadros