Artículo de investigación (Caso de estudio)
Sistemas Distribuidos Escalables
01 de Diciembre de 2015

Fury: Implementación de aplicaciones de alta demanda en Amazon AWS con Docker y repositorios Github

Daniel Andrés Valenzuela Suárez[1]

[1] *Departamento de Ingeniería Informática, Universidad de Santiago de Chile*

Correo electrónico: daniel.valenzuelas@usach.cl

Resumen

Ajustar el número de instancias (máquinas virtuales) que tiene asignada una aplicación en la nube, requiere un conocimiento detallado de la aplicación y, además, conocer el futuro dado que no se tiene una certeza de los eventos que sucederán. El resultado es que las arquitecturas de estas aplicaciones necesitan contar con un esquema controlado que permita la elasticidad en conceptos de escalabilidad, con el fin de asegurar el correcto funcionamiento de tal aplicación. La propuesta cuenta con un sistema reactivo en tiempo real que responde a los rangos previamente configurados, los cuales se detectan en las métricas asociadas. Las pruebas utilizan las **políticas estáticas a través de umbrales** con el fin de reaccionar ante el análisis en tiempo real de las métricas asociadas al escalamiento de la aplicación. La nueva propuesta de Fury, puede manejar cargas de trabajo de aplicaciones dinámicas de manera eficiente, reduciendo las pérdidas económicas y la insatisfacción del usuario.

Keywords: Scalability system, High disponibility, Deployment in Amazon AWS, IaaS, Cloud Computing, Fury, MercadoLibre

1. Introducción

La computación en la nube es una tecnología emergente que se está volviendo cada vez más y más popular. Esto se debe principalmente a su naturaleza elástica: las personas o empresas pueden adquirir y/o liberar recursos bajo demanda, y pagar sólo por los recursos que utilizan (pay-as-you-go). Estos recursos son por lo general en forma de instancias (máquinas virtuales). Las empresas pueden utilizar esta tecnología para diversos propósitos, tales como la ejecución de grandes procesamientos de datos, alojamiento de aplicaciones web, aplicaciones de intranet o bien copias de seguridad. Tres mercados principales están asociados a la computación en la nube:

- **Infraestructura-como-un-Servicio (IaaS):** designa la prestación de TI y recursos de la red, como el procesamiento, almacenamiento y ancho de banda, así como middleware de gestión. Los ejemplos son Amazon AWS (*Amazon Web Services*, 2015), RackSpace (*Rackspace*, 2015) o Google Compute Engine (*Google Compute Engine*, 2015).
- **Plataforma-como-un-Servicio (PaaS):** designa entornos y herramientas de programación soportados por los proveedores de la nube que pueden ser utilizados por los consumidores para construir y desplegar aplicaciones en la infraestructura cloud. Los ejemplos de PaaS incluyen Amazon Elastic Beanstalk (*Amazon Elastic Beanstalk*, 2015), Heroku (*Heroku*, 2015) o Force.com (*Force*, 2015).
- **Software-como-un-Servicio (SaaS):** designa aplicaciones alojadas de proveedores. Los herramientas Saas tenemos Google Apps (*Google Apps for Business*, 2015), Salesforce.com (*Salesforce.com*, 2015) o Microsoft Office 365 (*Microsoft Office 365*, 2015).

En este trabajo nos centraremos en la sección de infraestructura (IaaS). Para un mejor entendimiento de la infraestructura como un Servicio se describen los términos empleados en la investigación:

- **Proveedor:** es la entidad encargada de brindar el servicio de IaaS en nuestro caso en particular Amazon AWS, para hacer uso de sus recursos de manera controlada y dinámica.
- **Empresa:** es la entidad que solicita los servicios de IaaS, es la propietaria de la aplicación que se desea hospedar en la infraestructura.
- **Usuario:** es la persona que hace uso de la aplicación hospedada en la infraestructura, para nuestro caso las aplicaciones de MercadoLibre [MarketPlace, Payment, Security, Account, entre otros].

La elasticidad en la nube se define como el grado en que un sistema se adapta dinámicamente a la capacidad de carga de trabajo en el tiempo. En el contexto de la nube, la *capacidad* se refiere a la carga máxima de trabajo que un sistema puede manejar para cumplir sus objetivos de nivel de servicio. La elasticidad es una característica que diferencia a la computación en la nube de paradigmas de computación obsoletos, tales como Grid Computing. La adaptación dinámica de la capacidad, por ejemplo, mediante la alteración del uso de los recursos informáticos, para cumplir con una carga de trabajo variable se llama *computación elástica*. En general, una aplicación en la nube elástica o proceso tiene tres dimensiones elasticidad: *costo*, *calidad* y *recursos*, lo que permite aumentar y disminuir su costo, la calidad o los recursos disponibles, con el fin de dar cabida a los requisitos específicos.

La nube como bien la conocemos se puede dividir en 3 áreas distintas:

- **Las nubes públicas:** son nubes computacionales mantenidas y gestionadas por terceras personas no vinculadas con la organización. En este tipo de nubes tanto los datos como los procesos de varios clientes se mezclan en los servidores, sistemas de almacenamiento y otras infraestructuras de la nube.
- **Las nubes privadas:** son una buena opción para las empresas que necesitan alta protección de datos y ediciones a nivel de servicio. Las nubes privadas están en una infraestructura bajo demanda, gestionada para un solo cliente que controla qué aplicaciones debe ejecutarse y dónde.
- **Las nubes híbridas:** combinan los modelos de nubes públicas y privadas. Un usuario es propietario de unas partes y comparte otras, aunque de una manera controlada. Las nubes híbridas ofrecen la promesa del escalado, aprovisionada externamente, a demanda, pero añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos ambientes diferentes.

En este trabajo se propone una arquitectura de implementación de ambientes productivos en Amazon AWS que soporten alta disponibilidad en ambientes no controlados.

2. Estado del arte

Esta sección tiene como objetivo entregar las bases teóricas, conceptuales y empíricas que soportan cada desarrollo de este caso de estudio. En él se presentan las herramientas y plataformas que se emplearán para la realización de esta experiencia.

2.1. Técnicas de auto-escalamiento

Las actuales técnicas de auto-escalamiento abarcan diversas áreas de conocimiento entre las que se incluyen: políticas estáticas, aprendizaje por refuerzo, teoría de colas, teoría de control y análisis de series de tiempo. Cada una de estas categorías poseen muchos diversos métodos, los cuales pueden seguir un enfoque reactivo o proactivo (Lorido-Botrán, Miguel-Alonso, y Lozano, 2012).

Las técnicas reactivas hacen referencia a un conjunto de métodos que reaccionan en base a las decisiones referentes a los últimos valores obtenidos a partir de un conjunto de variables monitoreadas.

Las técnicas proactivas hacen referencia a un conjunto de métodos que reaccionan anticipadamente en base a las tendencias analizadas en las series de tiempos del conjunto de variables monitoreadas.

La falta de previsión de un enfoque reactivo afecta claramente el rendimiento del auto-escalamiento. El sistema podría no ser capaz de escalar proporcionalmente con el efecto Slashdot o ráfagas de tráfico repentinos resultantes de acciones del modelo de negocio en nuestro caso en particular de estudio el CyberDay.

En consecuencia de lo anterior, se estima para el caso de MercadoLibre a través de Fury un tiempo máximo de 5 minutos para generar una nueva instancia totalmente funcional (creación de la instancia, instalación de la imagen del sistema operativo, configuración de la arquitectura y deployment del repositorio en Github), y el efecto de una acción de auto-escalamiento podría llegar demasiado tarde en el caso de alta demanda no habitual. Por lo tanto, se requiere contar con un apoyo de métodos de predicción de eventuales alzas no habituales en la topología de la aplicación.

El análisis de series de tiempo cubre una amplia gama de métodos que siguen enfoques proactivos y parece ser un área de investigación prometedora. En esta categoría, se consideran todos los enfoques de auto-escalamiento que utilizan la historia pasada de una serie de tiempo en particular para predecir posibles cambios significativos.

- **Políticas estáticas a través de umbrales:** son métricas para determinar las acciones a seguir en caso de que se cumplan ciertos umbrales (rangos permitidos).
- **Aprendizaje por refuerzo:** es un área del aprendizaje automático inspirada en la psicología conductista, con el fin de determinar qué acciones se deben escoger en un entorno dado con el fin de maximizar alguna noción de recompensa o premio acumulado.
- **Teoría de colas:** es el estudio matemático de las colas o líneas de espera dentro de un sistema. Ésta teoría estudia factores como el tiempo de espera medio en las colas o la capacidad de trabajo del sistema sin que llegue a colapsarse.
- **Teoría de control:** es una área interdisciplinaria de la ingeniería y las matemáticas que se ocupa del comportamiento de los sistemas dinámicos con entradas, y cómo su comportamiento se modifica por la retroalimentación.
- **Análisis de series de tiempo:** es el análisis de una secuencia de datos, observaciones o valores, medidos en determinados momentos y ordenados cronológicamente.

2.2. Amazon AWS

Amazon Web Services (AWS abreviado) es una colección de servicios de computación en la nube (también llamados servicios web) que en conjunto conforman una plataforma de

computación en la nube, ofrecidas a través de Internet por Amazon.com. Es usado en aplicaciones populares como Dropbox, Foursquare, HootSuite. Es una de las ofertas internacionales más importantes de la computación en la nube compite y directamente contra servicios como Microsoft Azure y Google Cloud Platform. Es considerado como un pionero en este campo.

Amazon AWS se encuentra situado en 9 Regiones geográficas: USA Este (Norte de Virginia), USA Oeste (Norte de California), USA Oeste (Oregón), AWS GovCloud (USA), Sao Paulo (Brasil), Irlanda, Singapur, Tokio y Sydney. También hay un *GovCloud* en los USA. proporcionado para los clientes del Gobierno de USA. Cada región está totalmente contenida dentro de un solo país y todos sus datos y servicios permanecen dentro de la región designada (*Amazon Web Services*, 2015).

Cada región tiene múltiples *zonas de disponibilidad*, que son los diferentes centros de datos que proporcionan servicios de AWS. Las zonas de disponibilidad están aisladas unas de otras para evitar la propagación de cortes entre las zonas.

2.3. Docker

Docker es un proyecto de código abierto capaz de automatizar el despliegue de aplicaciones dentro de contenedores de software, proporcionándonos así una capa adicional de abstracción y automatización en el nivel de virtualización de sistema operativo sobre Linux. Docker hace uso de las utilidades de aislamiento de recursos del núcleo de linux como los **cgroups** y espacios de nombre del kernel, para permitir que *contenedores* independientes se ejecuten como una instancia única de Linux, evitando así la elevada sobrecarga en el arranque y mantenimiento de máquinas virtuales.

El soporte de espacios de nombres en el núcleo de Linux principalmente aisla una aplicación del entorno de operación, incluyendo árboles de procesos, IDs de usuario y sistemas de archivo montados, mientras que los cgroups proporcionan aislamiento de recursos, incluyendo la CPU, memoria, E/S de bloques y red. Desde la versión 0.9, Docker incluye la biblioteca **libcontainer** como su propia manera de utilizar directamente las facilidades de virtualización proporcionadas por el núcleo de linux, además de utilizar virtualización abstracta de interfaces a través de libvirt, LXC (contenedores de Linux) y systemd-nspawn (*Docker*, 2015).

Según la firma 451 Research, *Docker es una herramienta que puede empaquetar una aplicación y sus dependencias en un contenedor virtual, que a su vez puede correr en cualquier servidor de Linux. Esto proporciona flexibilidad y portabilidad allí donde la aplicación pueda ser ejecutada, como en una nube pública, una nube privada* (*451 Research*, 2015).

2.4. Github

GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago (*Github*, 2015).

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux (*Git*, 2015).

2.5. NewRelic

New Relic es un completo sistema de motorización de máquinas que permite tener un control en tiempo real de los recursos disponibles. Quizás sea uno de lo más conocidos y robustos para controlar aplicaciones webs, ahora lanza una interesante versión para hacer lo mismo pero de aplicaciones móviles Android e iOS.

New Relic cuenta con una sencilla instalación para monitorizar en tiempo real todo tipo de recursos de vuestras máquinas: procesos, carga de página, red, discos, etc, pudiendo fijar umbrales para el envío de alertas. Añadiendo el SDK de New Relic a nuestra aplicación, podemos conocer datos como el rendimiento de las conexiones de red a nuestros servidores o los **Crash Report** cuando falle (*NewRelic*, 2015).

Un tema que queda invisible cuando distribuimos una aplicación móvil es cuál será el rendimiento de las conexiones de red con nuestros servidores. Es decir, los fallos en las respuesta que puede dar la API que provee los datos a la aplicación móvil: excepciones, tiempos muertos, errores de los datos, etc. Con New Relic se puede conocer todos los tipos de mensajes HTTP que reciben nuestras aplicaciones, además de los tiempos de respuesta (dato importante para analizar si nuestra aplicación es lenta).

A través de los distintos paneles podemos ver las estadísticas que nos ofrece:

- **Conexiones HTTP:** tiempos de respuesta, mensajes de error, número de peticiones, etc.
- **Errores:** Crash Report cuando en la aplicación se produzca algún fallo tanto de ejecución como con la conexión a nuestra API.

- **Fijar alertas sobre umbrales de datos de referencia:** tiempos de respuesta, errores de autenticación, etc.
- **Estadísticas del rendimiento de la aplicación:** conociendo los tiempos de ejecución, el uso de la memoria o la velocidad de red.
- **Estadísticas de los usuarios:** según las distintas versiones de software del sistema operativo.

2.6. DataDog

DataDog es una herramienta SaaS basada en el seguimiento y análisis de la plataforma para la infraestructura de TI, operaciones y equipos de desarrollo. Reúne datos de servidores, bases de datos, aplicaciones, herramientas y servicios para presentar una visión unificada de las aplicaciones que se ejecutan en la nube a gran escala (*DataDog*, 2015).

DataDog está diseñado para poder agregar datos de aplicaciones en Chef, MySQL y Git, así como de otras herramientas especializadas y fuentes en la nube. Lo más interesante es que cubre todo el ciclo de vida de los cambios en el código dentro del desarrollo hasta las alertas de monitorización, con consistencia en el modelo de datos y en la experiencia end-to-end.

3. Arquitectura propuesta

Cuando se piensa en la arquitectura de una construcción, llegan a la mente muchos atributos distintos. En el nivel más sencillo, se considera la forma general de la estructura física. Pero, en realidad, la arquitectura es mucho más que eso, es la manera en la que los distintos componentes del edificio se integran para formar un todo cohesivo. Es la forma en la que la construcción se adapta a su ambiente y se integra a los demás edificios en la vecindad. Es el grado en el que el edificio cumple con su propósito y en el que satisface las necesidades del propietario. Es la sensación estética de la estructura y el modo en el que se combinan texturas, colores y materiales para crear la fachada en el exterior y el ambiente de vida en el interior. Es los pequeños detalles: diseño de las lámparas, tipo de piso, color de las cortinas, como vemos la lista es casi interminable.

La arquitectura no es el software operativo. Es una representación que permite 1) analizar la efectividad del diseño para cumplir los requerimientos establecidos, 2) considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y 3) reducir los riesgos asociados con la construcción del software.

Conforme las aplicaciones web se vuelven cada vez más importantes para la supervivencia y

el crecimiento empresarial, crece la necesidad de la administración de la configuración. ¿Por qué? Porque sin controles efectivos, los cambios inapropiados para una aplicación pueden conducir a: publicación no autorizada de información de productos nuevos, funcionalidad errónea o pobremente probada que frustra a los visitantes de un sitio web, huecos en la seguridad que ponen en peligro los sistemas internos de la compañía y otras consecuencias económicamente desagradables o incluso desastrosas.

Las estrategias generales para la administración de la configuración del software (ACS) son aplicables, pero las tácticas y las herramientas deben adaptarse para conformarse a la naturaleza única de las aplicaciones. Cuando se desarrollan tácticas para la administración de la configuración de una aplicación, deben considerarse cuatro conflictos (Dart, 1999).

- **Contenido:** Una aplicación típica contiene un arreglo muy amplio de contenido: texto, gráficos, complementos, guiones, archivos audio/vídeo, elementos de páginas activos, tablas, transmisión de datos, y muchos otros. El reto es organizar este mar de contenido en un conjunto racional de objetos de configuración) y luego establecer mecanismos de control de la configuración adecuados para dichos objetos. Por ejemplo, si un artículo de contenido cambia cada hora, tiene longevidad temporal. Los mecanismos de control para este artículo serán diferentes (menos formales) a los aplicados para un componente de formulario, que es un objeto permanente.

- **Personas:** Puesto que un porcentaje significativo de desarrollo aplicaciones continúa realizándose en una forma ad hoc, cualquier persona involucrada en la aplicación puede (y con frecuencia lo hace) crear contenido. Muchos creadores de contenido no tienen antecedentes de ingeniería de software y son completamente ajenos a las necesidades de la administración de la configuración. Como consecuencia, la aplicación crece y cambia en forma descontrolada.

- **Escalabilidad:** Las técnicas y los controles aplicados a una aplicación pequeña no escalan bien hacia arriba. No es raro que una aplicación simple crezca significativamente conforme se implementan interconexiones con sistemas de información, bases de datos, almacenes de datos y puertas a portales existentes. Conforme crecen el tamaño y la complejidad, pequeños cambios pueden tener efectos de largo alcance y no intencionados que pueden ser problemáticos. Por tanto, el rigor de los mecanismos de control de la configuración debe ser directamente proporcional a la escala de aplicación.

- **Políticas:** ¿Quién posee la aplicación? Esta pregunta se plantea en grandes y pequeñas empresas, y su respuesta tiene un impacto significativo sobre las actividades de administración y control. En algunas instancias, los desarrolladores web se alojan fuera del área de tecnología de la organización y crean potenciales dificultades de comunicación.

En consecuencia de las configuraciones que debe tener una aplicación y de la variabilidad con la cual puede interactuar dados eventos programados y no programados es que se propone una arquitectura para las aplicaciones tanto Frontend como Backend a través de Fury, el cual es una aplicación propietaria de MercadoLibre para inicializar una aplicación en ambiente productivo a través de Amazon AWS, esta funcionalidad se realiza a través de la API pública que ofrece Amazon y de los archivos de configuración de Docker que contienen las configuraciones de la imagen del sistema operativo y un conjunto programado de aplicaciones que se deben instalar para dejar operativa la instancia. Además se cuenta con el soporte directo en Github que cuenta con la aplicación propiamente tal, además de la configuración interna que poseen las instancias.

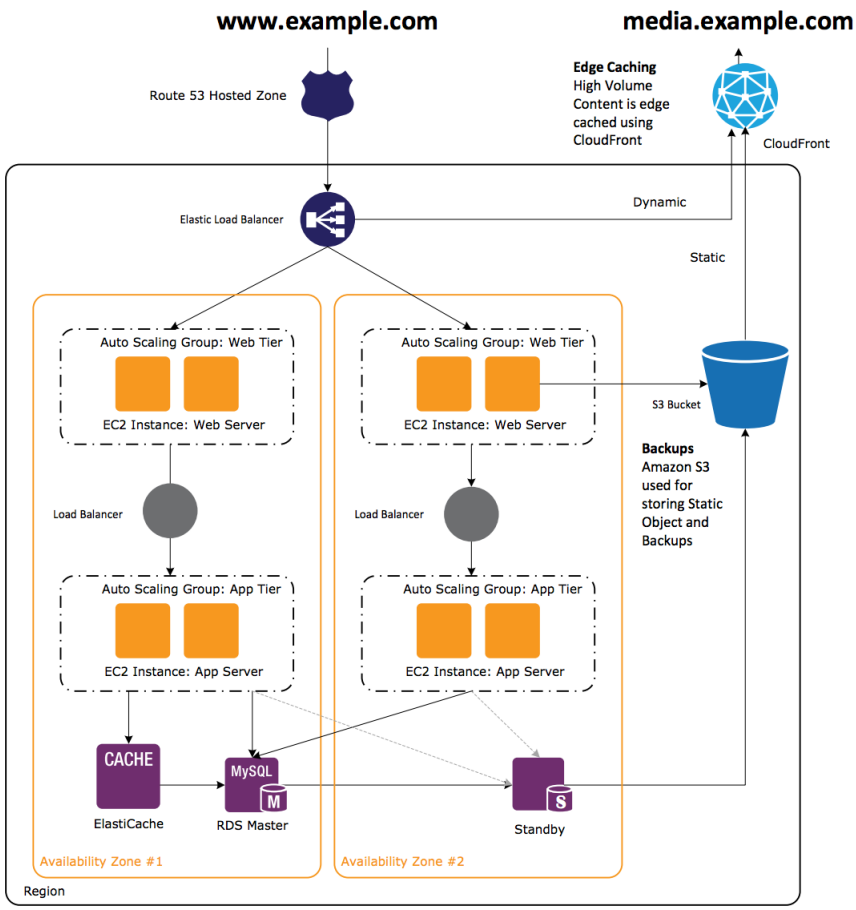

Figura 1: Arquitectura propuesta para auto-escalamiento

Esta VPC (Virtual Private Cloud) cuenta con enrutadores de tráficos, balanceadores de carga, instancias de procesamiento (servicio para Backend y de interfaz para Frontend), almacenamiento en disco, almacenamiento CDN, memoria caché y protocolos de seguridad, tal como se visualiza en la figura 1. Esta arquitectura se encuentra diseñada con 3 capas de instancias: las instancias activas (que se encuentran operativas), las instancias inactivas (que se encuentran configuradas pero apagadas) y las instancias de escalabilidada (lo que escalará según corresponda).

4. Simulación con métricas

Para el proceso de simulación, se desarrolló un ambiente gráfico a través de componentes web, con el fin de analizar el comportamiento de la arquitectura propuesta ante eventuales variaciones repentinas de las métricas analizadas (CPU, Latencia, Memoria y Tiempo de respuesta).

Este proceso de simulación se encuentra en el área de protocolos reactivos de auto-escalamiento, con el propósito de optimizar las aplicaciones en las métricas asociadas y, por otra parte, rentabilizar económicamente el uso correcto del IaaS que requiere la aplicación.

Las métricas utilizadas para esta simulación enfocan las 4 grandes características que cualquier aplicación necesita contar a la hora de tener sus servicios en un ambiente de alta demanda, tal y como se muestra en la figura 2, estas métricas sirven tanto en aplicaciones Frontend (lado cliente) como Backend (lado servidor).

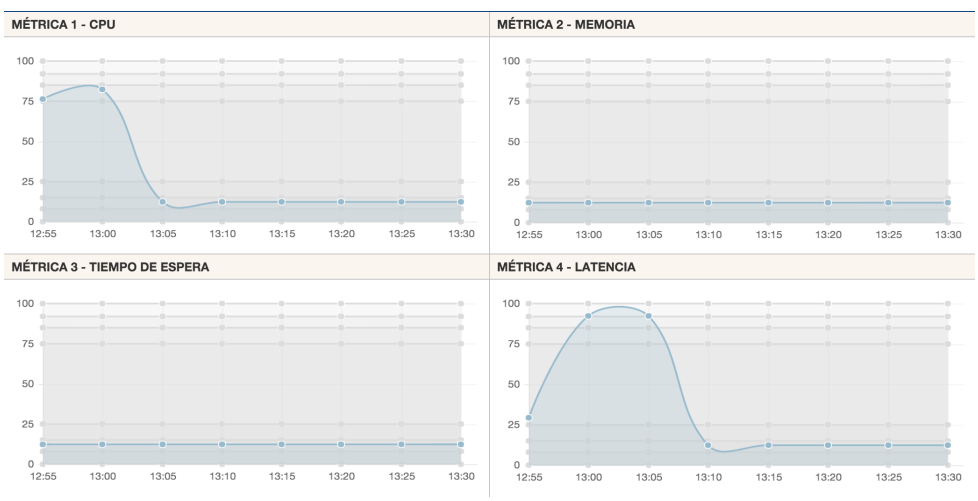

Figura 2: Métricas analizadas por el simulador

El simulador se desarrollo con un conjunto de configuraciones básicas para la ejecución del protocolo reactivo anteriormente mencionado, para ello se cuenta con:

- Tiempo de ejecución para la obtención de las métricas y su posterior análisis, por defecto se dejo configurado cada 5 minutos debido a que el proceso de escalamiento en Amazon AWS a través de API para MercadoLibre tienen una fluctuación entre 3 a 4 minutos.
- Número mínimo y máximo de instancias permitidas en la VPC dada la arquitectura propuesta, esto permite adicionalmente de la escalabilidad, elasticidad a nivel de crecimiento cuando corresponda y en caso de contar con bajos índices en cada una de las métricas, ir disminuyendo su configuración en tiempo real.
- Rangos de elasticidad con las configuraciones de auto-escalamiento, esto permite que según sea el comportamiento en tiempo real de la métrica Fury se encargue de configurar la arquitectura para optimizar los recursos y conseguir un beneficio económico de ahorro (pagar solamente si es necesario).

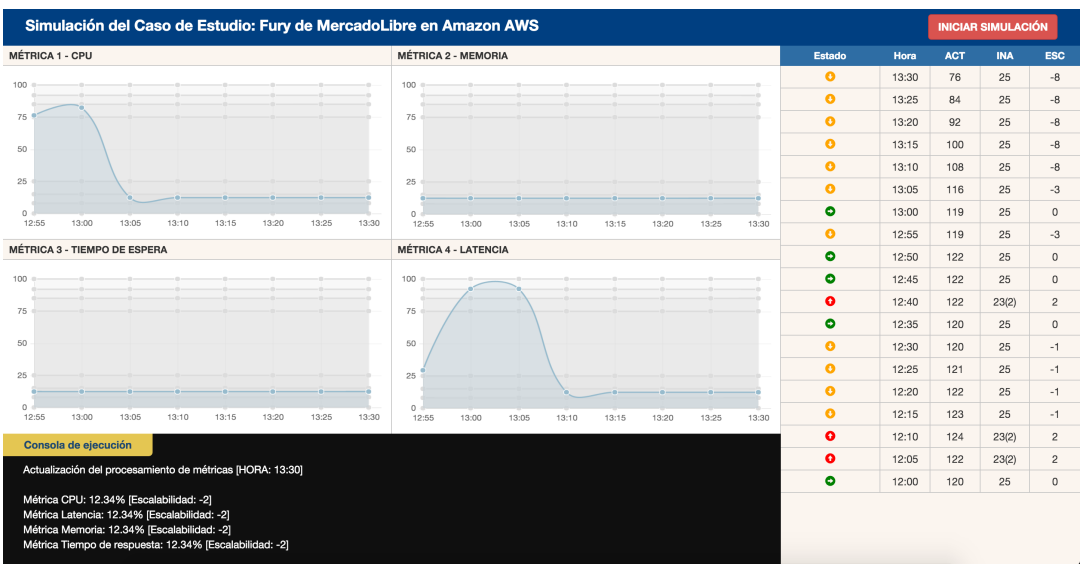

Figura 3: Aplicación que simula las métricas de auto-escalamiento

Como se visualiza en la figura 3, la aplicación desarrollada presenta 3 secciones, la primera de ellas muestra las métricas obtenidas para su posterior análisis de auto-escalamiento con elasticidad; la segunda muestra la sección de la consola de ejecución que visualiza los cálculos obtenidos respecto a los rangos asociados a cada una de las métricas y, por último, se muestra el historial de la arquitectura en tiempo real de ejecución y su composición.

Como se logra ver en la figura 4, esta sección muestra el comportamiento de la arquitectura a nivel de auto-escalamiento y elasticidad según el análisis en tiempo real de cada una de las métricas analizadas.

Estado	Hora	ACT	INA	ESC
↓	13:30	76	25	-8
↓	13:25	84	25	-8
↓	13:20	92	25	-8
↓	13:15	100	25	-8
↓	13:10	108	25	-8
↓	13:05	116	25	-3
→	13:00	119	25	0
↓	12:55	119	25	-3
→	12:50	122	25	0
→	12:45	122	25	0
↑	12:40	122	23(2)	2
→	12:35	120	25	0
↓	12:30	120	25	-1
↓	12:25	121	25	-1
↓	12:20	122	25	-1
↓	12:15	123	25	-1
↑	12:10	124	23(2)	2
↑	12:05	122	23(2)	2
→	12:00	120	25	0

Figura 4: Historial de la arquitectura

5. Conclusiones

La simulación de la arquitectura propuesta presenta un comportamiento estable en relación a la toma de decisiones respecto al proceso de auto-escalamiento reactivo, realizando según corresponda los 3 tipos de acciones que puedo realizar: Upgrade, Downgrade o bien seguir tal como se encuentra.

Al contar con una conexión directa con la API de Amazon AWS, el proceso de cambiar la arquitectura en relación a las máquinas Inactivas a Activas, es una evento eficiente que en poco menos de 5 segundos ya se encontraría ya operativa la nueva instancia, dejando de lado los posibles acontecimientos que se puedan ver reflejados por el alza de los índices en la métrica.

El proyecto se encuentra disponible en el repositorio Github:
`https://github.com/softcrum/research_fury-scalability`

6. Discusión científica

Contar con una aplicación que realice toda la administración y configuración en tiempo real de las aplicaciones de alta demanda, optimiza los tiempos de respuesta ante eventos no programados y realiza las acciones pertinentes para su estabilidad en la ejecución.

Como trabajo futuro, se propone realizar una optimización a esta aplicación abordando temas de presupuesto y rentabilidad en el apagado de las instancias en Amazon AWS; incorporar las métricas de NewRelic y DataDog a través de APIs con el fin de obtener las métricas en tiempo real y no como un proceso en bruto para realizar una simulación; incorporar un tiempo mínimo y máximo de estabilidad o sobrecarga para la realización del proceso de auto-escalamiento, en estos momentos la aplicación realiza la ejecución de manera secuencial según el análisis de las métricas.

Agradecimientos

Este trabajo se encuentra con el apoyo de MercadoLibre y sus área de IT y sistemas, con el fin de optimizar las tecnologías ya existentes dentro de la empresa.

Referencias

451 research. (2015). Descargado 2015-11-19, de http://www.linux.com/news/enterprise/cloud-computing/731454

Amazon elastic beanstalk. (2015). Descargado 2015-11-19, de http://aws.amazon.com/elasticbeanstalk/

Amazon web services. (2015). Descargado 2015-11-19, de http://aws.amazon.com/ec2/

Dart, S. (1999). Change management: Containing the web crisis. En *Ics workshop on web engineering.*

Datadog. (2015). Descargado 2015-11-19, de http://www.datadog.com/

Docker. (2015). Descargado 2015-11-19, de http://www.docker.com/

Force. (2015). Descargado 2015-11-19, de http://www.force.com/

Git. (2015). Descargado 2015-11-19, de http://www.git-scm.com/

Github. (2015). Descargado 2015-11-19, de http://www.github.com/

Google apps for business. (2015). Descargado 2015-11-19, de https://www.google.com/work/apps/business/

Google compute engine. (2015). Descargado 2015-11-19, de http://cloud.google.com/

Heroku. (2015). Descargado 2015-11-19, de http://www.heroku.com/

Lorido-Botrán, T., Miguel-Alonso, J., y Lozano, J. A. (2012). Auto-scaling techniques for elastic applications in cloud environments.

Microsoft office 365. (2015). Descargado 2015-11-19, de http://www.microsoft.com/en-us/office365/online-software.aspx

Newrelic. (2015). Descargado 2015-11-19, de http://www.newrelic.com/

Rackspace. (2015). Descargado 2015-11-19, de http://www.rackspace.com/

Salesforce.com. (2015). Descargado 2015-11-19, de http://www.salesforce.com/