SG SOFTWARE GURU

NO.52 | CONOCIMIENTO EN PRÁCTICA
www.sg.com.mx

NOVEDADES
OCULUS +
LEAP MOTION
PAG. 12

ESCALA

USANDO ARQUITECTURAS PARA LA NUBE

ABIZTAR
LEARNING TECHNOLOGIES

hace una diferencia en la educación en México

Al inscribirte a un Ambiente de Aprendizaje Abiztar, no sólo obtienes una capacitación más avanzada y eficiente a la que has probado hasta el momento.

Al inscribirte a un Ambiente de Aprendizaje Abiztar, ayudas a educar a un niño en situación de calle por medio de la fundación Ednica I.A.P. (ednica.org.mx).

Nuestro compromiso contigo no termina con un curso, termina con el éxito de tus proyectos.

Al concluir la fase presencial de tu capacitación recibes un dibujo hecho por el niño que ayudas, en agradecimiento por tu aportación.

ednica
educación con niños, niñas, adolescentes y jóvenes en situación de calle

Más sobre los ambientes en: http://abiztar.com.mx/SG

+52 (55) 5594 6411 / cursos@abiztar.com.mx / www.abiztar.com.mx

70 AÑOS
DE COMPROMISO CON LA LIBERTAD Y LA EXCELENCIA

Cecilia Ortiz
PROFESORA DE DIRECCIÓN DE MERCADOTECNIA

Posgrados

- MBA-Maestría en Administración
- Maestría en Administración de Riesgos
- Maestría en Ciencia de Datos
- Maestría en Ciencias en Computación
- Maestría en Contaduría
- Maestría en Derechos Humanos y Garantías
- Maestría en Dirección Internacional
- Maestría en Economía Aplicada
- Executive MBA-Maestría en Dirección de Empresas
- Maestría en Finanzas
- Maestría en Mercadotecnia
- Maestría en Políticas Públicas
- Maestría en Tecnologías de Información y Administración
- Maestría en Teoría Económica
- Doctorado en Economía

ITAM Posgrados

PREGUNTA POR NUESTRAS SESIONES INFORMATIVAS SEGÚN EL PROGRAMA DE TU INTERÉS.

Av. Camino a Santa Teresa No. 930 Col. Héroes de Padierna, C.P. 10700, Ciudad de México, México.
Tel: (55) 5628 4000 ext. 2612, 01 800 000 ITAM,
posgrados@itam.mx, www.posgrados.itam.mx
Síguenos en: Posgrados ITAM PosgradosITAM

MBA-Maestría en Administración, R.V.O.E. 2001, 01/10/01; Dirección Internacional, R.V.O.E. 944005 13/06/94; Economía Aplicada, clave 2015; Finanzas R.V.O.E. 944195 14/11/94; Tecnologías de Información y Administración, R.V.O.E. 984028 15/04/96; Doctorado en Economía, R.V.O.E. 88820 09/03/01. Reconocimiento de validez oficial mediante Decreto Presidencial publicado en el Diario Oficial de la Federación el 19 de enero de 1963: Administración de Riesgos, clave 2002; Contaduría, clave 2012; Executive MBA-Maestría en Dirección de Empresas, clave 2008; Mercadotecnia, clave 2011; Maestría en Políticas Públicas, clave 2014; Teoría Económica, clave 2001. Maestría en Ciencia de Datos, Maestría en Ciencias en Computación y Maestría en Derechos Humanos y Garantías: clave 2012.

Transformando México

Foro Amiti 2016

Transformando a la industria, a la sociedad, al país a través de la tecnología

13 de octubre en el Centro Banamex

¡Reserva en tu agenda!

foroamiti.com

SG VIRTUAL CONFERENCE

11ª edición

Construyamos mejor software sin fronteras

Más de 20 conferencias virtuales sobre herramientas, tecnologías y mejores prácticas para el desarrollo de software, impartidas por ponentes internacionales

26 de octubre 2016

EVENTO GRATUITO

Sede: Todo el mundo

¡Regístrate ahora!

http://sg.com.mx/sgvirtual

Mayores informes

sgvirtual@sg.com.mx

Tel: +52(55) 5239-5502

SG SoftwareGuru

@RevistaSG

SG®
SOFTWARE GURU

NO.52 | CONOCIMIENTO EN PRÁCTICA
www.sg.com.mx

EN PORTADA

ARQUITECTURAS DE SOFTWARE PARA LA NUBE

Conoce las principales tendencias arquitectónicas relacionadas con la nube y el big data.

I INDUSTRIA Y EMPRESAS

T HERRAMIENTAS Y TECNOLOGÍAS

P PRÁCTICAS

C COLUMNAS

V VOCES

O EN CADA NÚMERO

A la evolución no le importa si estás lista(o) para ella.

En nuestra profesión estamos acostumbrados al cambio continuo; siempre ha habido y habrá nuevas tecnologías, herramientas, proveedores, y mejores prácticas. En realidad, más que acostumbrados estamos curtidos por el cambio. Una de las consecuencias de esto es que conforme pasan los años tendemos a hacernos escépticos de las tendencias. El inconveniente de esto es que corremos el riesgo de ver venir la ola hacia nosotros e ignorarla hasta que es demasiado tarde.

La misión de Software Guru es llevar y mantener a la vanguardia a los profesionistas de software de nuestra región. Para nosotros, esto consiste no solo en decirte que ahí viene la ola, sino en también darte una tabla y guiarte hacia la ola para que la surfees —y de paso echarte porras mientras lo haces.

Es por ello que en este número de SG nos enfocamos en platicar sobre arquitecturas modernas de software, moldeadas por la incursión del cómputo en la nube y la transformación digital. Consideramos que es un tema clave, pero al que desgraciadamente se le está dando poca atención en nuestra región. Confiamos en que los lectores de SG comprenderán la importancia de este tema y lo enfrentarán con la dedicación que merece para navegar esta ola con gran éxito.

El equipo de Software Guru

SG es posible gracias a la colaboración de

Dirección Editorial **Pedro Galván** | Dirección de Operaciones **Mara Ruvalcaba** | Dirección Comercial **Claudia Perea**
Coordinación Editorial **Susana Tamayo** | Arte y Diseño **Oscar Sámano** | Suscripciones **Mariana Torres**
Consejo Editorial: Luis Daniel Soto | Gunnar Wolf | Luis Vinicio León | Hanna Oktaba
Ariel Jatuff | Emilio Osorio | Gloria Quintanilla | Jorge Valdés

COLABORADORES EN ESTA EDICIÓN
Ernesto Riestra, Pedro Ramírez, Edith Gómez, Benjamín Vera-Tudela, Cyrille LeClerc, Paul Biggar, Aad van Schetsen, Humberto Cervantes, Misael León, Roselyn C. Piñango, Omar Sánchez.

EQUIPO SG
Coordinación de servicio **Yoloxochitl Juárez** | SG Campus y proyectos especiales **Hilda Ramírez** | Business Development **Erika Rivero**
Developer Relations **Luis Sánchez** | Servicios online **Ivett Sánchez** | Alianzas **Lorena Mireles** | Contenidos **Ana Loyo**
Contacto: info@sg.com.mx

SG Software Guru es una publicación trimestral editada por Brainworx, S.A. de C.V., San Francisco 238 Altos. Col. Del Valle. *Reserva de Derechos al Uso Exclusivo: En trámite. ISSN: 1870-0888. Registro Postal: PP15-5106. Distribuido por Sepomex.*

Noticias

OPENSTACK HACKATHON GUADALAJARA 1

Del 9 al 11 de septiembre se realizó el OpenStack Hackathon en la ciudad de Guadalajara, México. En él participaron más de 100 desarrolladores que a lo largo de 3 días construyeron soluciones innovadoras utilizando la plataforma de código abierto OpenStack para gestionar infraestructura de cómputo. El equipo Sensemaya resultó ganador, construyendo una aplicación que permite a los usuarios tener su expediente clínico electrónico a la mano de manera que pueda ser fácilmente accesible en caso de una emergencia médica. Como ganadores, los integrantes del equipo se hicieron acreedores a un viaje al OpenStack Summit que se realizará en Barcelona en octubre de este año. Este fue apenas el segundo hackathon de OpenStack que se realiza en el mundo. Fue organizado por la comunidad local de desarrolladores OpenStack con el patrocinio de Intel, Red Hat, Tecnológico de Monterrey Campus GDL, y el apoyo de Software Guru.

CIRCUITO TECNOLÓGICO AVANTARE 2

El 27 de septiembre se realizó en Ciudad de México una edición más del Circuito Tecnológico impulsado por PROSOFT 3.0, con el propósito de compartir el conocimiento que Empresas, Organizaciones y Gobierno tienen en TI. Esta edición fue organizada por la empresa Avantare y se enfocó en discutir Modelos de Calidad e Innovación en las empresas. Se contó con la participación de 5 empresas acreditadas en CMMI, quienes compartieron sus experiencias y coincidieron en que a pesar de que el camino de obtener una acreditación no es sencillo, vale la pena afrontar el reto pues los beneficios son mayores que el sufrimiento. Vale la pena mencionar que México es uno de los países que mayor crecimiento tuvo en 2015 en el número de empresas acreditadas en CMMI.

KLOUD CAMP 2016 3

"Klouduniverse" fue el alias del Kloud Camp 2016, evento realizado en Ciudad de México por Kio Networks con el objetivo de promover el cómputo en la nube. Esta fue la sexta edición de este evento, al que asistieron más de 5 mil personas y en el que destacaron conferencias de personalidades como María Asunción Aramburuzabala, Jason Silva y Andrés Velázquez, entre otros. Como parte del evento se dio a conocer The Garage, una iniciativa de Kio Networks para apoyar a empresas de base tecnológica, en donde las startups participantes tuvieron oportunidad de realizar su "pitch" para presentarse. Las startups premiadas fueron Gestionix, una plataforma en la nube para administrar empresas, desde la cobranza hasta los inventarios; y Karma Pulse, una suite de herramientas de análisis de redes sociales y data web.

TECHSTARS LATAM SUMMIT 4

La comunidad de organizadores y colaboradores de Startup Weekend en Latinoamérica se reunió en el Techstars Latam Summit realizado en Santiago de Chile del 15 al 17 de septiembre. Durante el evento se exploraron y compartieron ideas para continuar impulsando el emprendimiento tecnológico en nuestra región. También sirvió para fortalecer los lazos entre colegas de los distintos países y darnos cuenta de que enfrentamos retos muy similares. El evento fue organizado por Techstars, la aceleradora global de startups propietaria de Startup Weekend. También durante septiembre, Techstars realizó una serie de sesiones informativas en Latinoamérica para conocer startups locales y difundir su programa de aceleración en nuestra región. Las sesiones informativas se realizaron en Ciudad de México, Bogotá, Buenos Aires, Sao Paulo, Lima y concluyeron en Santiago empatando con las actividades del Techstars Latam Summit.

SGNEXT

El 30 y 31 de agosto se realizó en Ciudad de México la primera edición del congreso SG Next. Compartimos un poco de lo que fue este evento lleno de descubrimiento, aprendizaje y convivio entre colegas.

PARTICIPACIÓN

Asistieron 426 personas de 14 distintos estados de la República Mexicana. Los participantes de la empresa Code Services en Ensenada, Baja California fueron quienes recorrieron más distancia para asistir al evento. La empresa con mayor representación de fue el Instituto Mexicano del Petróleo con 16 asistentes, seguidos por OCC Mundial con 12. Reconocemos el interés de estas organizaciones de mantener a sus colaboradores a la vanguardia.

CONTENIDOS

A lo largo de sus dos días, SG Next contó con una gran variedad de sesiones que incluyeron temas como internet de las cosas (IoT), cómputo visual, User Experience Design, bots, machine learning, DevOps, seguridad de la información y testing automatizado. Realmente había contenido para una gran variedad de roles e intereses dentro del desarrollo de software. La conferencia mejor evaluada fue "Serverless" de Obie Fernández, seguida por "Desarrollo Seguro de Aplicaciones" de Héctor Paredes.

Las presentaciones están disponibles en http://slideshare.net/RevistaSG

ACTIVIDAD EN REDES SOCIALES

#sgnext tuvo gran actividad en redes sociales. Durante el evento 318 usuarios distintos acumularon cerca de 1,500 tweets, con un índice positivo de más de 98%. Estos mensajes alcanzaron a 337,863 cuentas distintas para un potencial total de más de 10 millones de impresiones.

SHOWCASE

Una de las novedades de SG Next fue que se contó con un espacio dedicado a dar a conocer nuevos proyectos y emprendimientos locales relacionados con tecnologías como internet de las cosas y cómputo visual. Esto fue un gran éxito ya que a través de la interacción los participantes identificaron oportunidades de aplicación de estas tecnologías en sus propios proyectos o empresas.

PATROCINADORES

SG Next contó con la participación de las empresas que tienen mayor compromiso con los desarrolladores de software en nuestra región. Agradecemos la participación de Microsoft, Red Hat, Baufest, SAP, Abiztar, Four Js, e-Quallity, Meltsan Solutions, Metagraphos, GFT, Tacit Knowledge, Qualtop, Softtek, Cisco y Eventto.

Adicionalmente, SG Next contó con el apoyo de la Secretaría de Economía, por medio del programa PROSOFT 3.0.

La próxima edición de SG Next se está planeando para junio 2017 en Ciudad de México. En cuanto tengamos la fecha final te la haremos saber. ¡Nos vemos pronto!

DevDay 4 <WOMEN>

ÚNETE A LAS MUJERES QUE DESARROLLAN
SOFTWARE GRANDIOSO

PRÓXIMA EDICIÓN

2 DE DICIEMBRE 2016, CIUDAD DE MÉXICO

http://devday4w.com

Pioneras en la Preocupación por la **Calidad en Ingeniería de Software**

MIS RECUERDOS DEDICADOS A LAS JÓVENES INVOLUCRADAS EN TI

Por Hanna Oktaba

La Dra. Hanna Oktaba es profesora de la UNAM y su objetivo principal es generar conocimiento a través de la creación y promoción de estándares.
@hannaoktaba

Hace unos días mi amiga Gloria Quintanilla me mandó una foto vieja encontrada en el baúl de su mamá. Esta foto ha despertado muchos recuerdos que quiero compartir con ustedes por dos razones: la primera es que se trata de una foto histórica de 1997, que presenta al grupo que fue uno de los pioneros al hacer algo por la calidad de software en México. La segunda razón es que son puras mujeres.

En aquel entonces tuve la oportunidad de disfrutar por primera vez el año sabático en la UNAM. Normalmente para pasar el año sabático los académicos escogen a una universidad para colaborar con otros investigadores. Pero en mi informe de sabático encuentro que efectivamente tuve una estancia académica en el Departamento de Electrónica del Politécnico de Milán (junio 1997 – julio 1997). Durante la cual me dediqué a estudiar el concepto de Proceso de Software y sus distintas formas de modelado. En particular, estudié el modelo de Personal Software Process de Watts Humphrey, que en ese entonces era casi desconocido en México.

Sin embargo, el resto de mi sabático (agosto 1997 – mayo 1998) tuve una estancia de colaboración en la empresa Tecnosys, en la ciudad de México, durante ese tiempo:

- Participé en la definición e implantación del sistema de calidad en la empresa el cual la llevó a la obtención del certificado de ISO9001 en diciembre de 1997.
- Ofrecí dos cursos para el personal de la empresa sobre el "Panorama de Tecnología Orientada a Objetos" y "Proceso Personal de Software".
- Desarrollé una propuesta del programa de métricas para Tecnosys.
- Participé en la definición del modelo conceptual de una biblioteca de componentes reutilizables para la empresa.
- Apoyé a la gerente de calidad en la coordinación de su grupo de trabajo.

Gloria, que aparece en la foto a mi lado izquierdo, fue justamente esa gerente de calidad que tuvo la idea de invitarme a ayudarle a implementar el sistema de calidad basado en ISO 9001 en una empresa de desarrollo de software recién comprada por IBM de México, cuyo desafío adicional fue adoptar los procesos de desarrollo de la propia IBM. El reto fue muy grande. Nuestro equipo de trabajo lo completaban Angélica Su Ramos y Cecilia Montero Mejía (primera y segunda de la izquierda en la foto), ambas fueron mis ex alumnas de la Maestría en Ciencias de la Computación de la UNAM, y Mariana Pérez-Vargas de Tecnosys (de mi lado derecho en la foto). También, incorporamos a mi ex alumno de maestría Carlos Pérez Escobar, que no aparece en la foto, para no desdeñar la visión masculina ;) Una proporción parecida se repitió en el equipo que definió MoProSoft: fuimos ocho mujeres y tres hombres.

Para documentar el sistema de calidad con todos los elementos requeridos, Gloria encontró una herramienta irlandesa llamada Aimware, pionera en el mercado. Invitamos a Eamon McGuiness, el dueño de la empresa, para que nos convenciera y capacitara en su uso.

Contamos con cinco meses para "traducir" los requisitos de ISO 9001:1995, enfocados en lenguaje de empresas manufactureras, a los conceptos de los proyectos de desarrollo de software. Lo más difícil, que se me grabó en la memoria, era entender que en la manufactura hay que cuidar la calidad de cada copia producida a partir de un prototipo del producto, pero en el desarrollo de software lo importante es cuidar la calidad durante el desarrollo del prototipo, porque las copias salen con la misma calidad sin problema.

Cuando ya teníamos definidos todos los elementos del sistema de calidad con sus políticas, procesos y procedimientos documentados en Aimware, empezamos a buscar un auditor para revisar nuestra propuesta. La tarea no fue fácil porque no se contaba con experiencias de auditorías de ISO 9001 en las empresas de desarrollo de software (donde los productos son intangibles) y menos con la documentación en forma electrónica —todo se documentaba en papel. La que aparece segunda de la derecha en la foto es Margarita Santos, la única auditora que aceptó el reto. Apreciamos mucho su apertura para comprender las particularidades del desarrollo de software, como por ejemplo, que los insumos provienen del conocimiento humano, y su aceptación de que el manual de calidad estuviera albergado en un sistema de software.

Toda esta aventura se terminó con una auditoría exitosa en diciembre de 1997 y dio pie a que en el año siguiente con esta base trabajáramos en la adopción de SW-CMM nivel 2 y 3. Como efecto lateral, desde septiembre de 1997 en Tecnosys empezamos reuniones mensuales, de lo que llamamos "Círculo de Calidad de Software", en las cuales invitábamos a todos los interesados en temas de calidad en ingeniería de software. En las reuniones se hacían presentaciones de los modelos y estándares como CMM, ISO/IEC 15504, RUP entre otros. Su éxito fue tal que en 1999 fundamos la Asociación Mexicana para la Calidad en Ingeniería de Software (AMCIS).

Pero, ¿qué ha pasado con las mujeres guapas que aparecen en la foto?, además de haber logrado varios éxitos profesionales y contribuir con un granito de arena en temas de calidad para la industria de TI en México, son felices madres de familia que han logrado conjuntar la vida profesional con la familiar. Gloria tuvo una participación crucial en el diseño de MoProSoft y MAAGTIC y es consultora en dirección de proyectos, planeación estratégica y gobierno para resultados, con una experiencia muy importante para el gobierno de Ecuador. Mariana es SCAMPI Lead Appraiser y socia fundadora de la consultora Avantare, que es líder en el mercado mexicano bajo su dirección. Angélica es coautora de MoProSoft y consultora experta en modelos y estándares de TI, actualmente está dirigiendo proyectos de TI dentro del sector financiero. Cecilia fue la primera evaluadora de América Latina reconocida por el Software Engineering Institute como Lead Appraiser de SW-CMM, posteriormente se convirtió en la primera instructora de CMMI certificada en nuestro país. Actualmente imparte cursos en México y varios países de habla hispana y también se dedica a la consultoría.

A manera de dedicatoria a nuevas generaciones de mujeres involucradas en TI quiero decirles que seguro están igual o mejor preparadas que nosotras en su momento. El avance en la tecnología y las opciones de acceso al conocimiento y al emprendimiento son oportunidades que sin duda las mujeres jóvenes deben aprovechar para llevar adelante iniciativas, asumir el papel de liderazgo a nivel profesional y, a la par, cumplir sus sueños personales.

1 APACHE FLINK AMENAZA A SPARK

El mundo del big data se está moviendo a un ritmo tal, que a una tecnología con apenas un par de años de edad se le pueden empezar a ver las arrugas. Tal es el caso de Spark, un motor de procesamiento de datos de alto desempeño del que ya hemos hablado en las páginas de SG, y que hasta hace unos meses era la tecnología más candente del big data. Hoy, parece que ese lugar lo está tomando Flink, otro proyecto de la Fundación Apache.

Tanto Spark como Flink son frameworks open source para el procesamiento distribuido de datos a alta velocidad, y ambos representan una mejora significativa respecto a MapReduce. Ambos se pueden utilizar para analizar corrientes de datos (*stream processing*), aunque la forma en que lo hacen es distinta. Digamos que Spark procesa los datos en modo batch, pero lo hace con tal frecuencia y velocidad que se aproxima a tiempo real, mientras que Flink sí procesa las corrientes como tales, procesando cada dato inmediatamente conforme es recibido. Esto le da a Flink una muy ligera ventaja de velocidad. Otra ventaja de Flink es que permite manejar la memoria de forma explícita, evitando así los picos generados por tareas de recolección de basura.

Actualmente, Spark tiene un ecosistema más amplio y maduro que Flink, que apenas hizo disponible su versión 1.0 en marzo de este año. Así que a estas alturas todavía sería arriesgado poner en producción algo con Flink, pero sin duda vale la pena mantenerlo en el radar y ver cómo evoluciona.

2 SWIFT 3 ENDEREZA EL CAMINO A COSTA DE RETROCOMPATIBILIDAD

Apple liberó la versión 3 del lenguaje de programación Swift. Esta es la primera versión mayor de Swift posterior a la que se hiciera disponible como software libre. Pero lo que más llama la atención es que no es retrocompatible con versiones anteriores. Al parecer, la empresa de la manzana no estaba satisfecha con cómo se habían implementado varios detalles, entre ellos la traducción de código de Objective-C a Swift. La buena noticia es que XCode 8 tiene una herramienta para migrar código de Swift 2.2 y 2.3 a Swift 3.

3 CEYLON YA SOPORTA ANDROID, NPM Y WILDFLY

Ceylon es un lenguaje de programación open source para la máquina virtual de Java (JVM) del que también ya hemos hablado en las páginas y eventos de SG. La novedad es que Ceylon recién lanzó su versión 1.3, que se ve bastante bien. Entre sus principales características están:

- Programación de aplicaciones Android nativas.
- Capacidad de importar y publicar módulos en npm.
- Desarrollo de microservicios con WildFly Swarm.
- Plugin de IDE para IntelliJ IDEA y Android Studio.

Si te interesa conocer más sobre Ceylon, en la próxima edición de SG Virtual tendremos una plática al respecto.

4 GOOGLE VR SDK SE GRADÚA DEL BETA

Como probablemente ya te habrás dado cuenta, Google le está apostando fuertemente a la realidad virtual, y justo está lanzando su plataforma Daydream que incluye una nueva línea de smartphones, visores y arreglos de cámara 3D. Google VR SDK es el kit para desarrollar aplicaciones de VR para estos dispositivos; que se dio a conocer en el Google I/O de este año y a partir de entonces estuvo disponible en beta, pero ya se graduó a 1.0. El SDK resuelve aspectos comunes en el desarrollo de apps VR tales como el reconocimiento de interacciones del usuario, manejo de audio espacial y reproyección asíncrona. Una gran noticia es que el Google VR SDK se puede usar tanto con Unity como con Unreal. Así que si utilizas estos motores de juegos, te será mucho más fácil comenzar a construir apps de VR.

Gartner CIO & IT Executive Latin America Summit

14 - 17 noviembre 2016
Cancún, México
www.gartner.com/americas/cio
Tels: +52 55 9171 1714

Gartner Summits

Algunas sesiones destacadas:

- Agenda 2017 del CIO en América Latina
- Innovación a partir de la analítica
- Estado de la seguridad de la nube en 2016
- Tendencias móviles y de ***wearables*** en el 2020
- De socio a aliado de confianza, ¿cómo seguir siendo relevante como CIO?

Sesión Magistral Especial

Innovación en América Latina:
El Surgimiento de los Nuevos Ecosistemas Digitales

Dr. Raúl Katz

Director del Centro para la Transformación del Negocio Digital de gA y Director del Centro de Investigación en Estrategia Empresarial del *Columbia Institute for Tele-Information, Columbia University.*

Entre nuestros patrocinadores están:

Premier

Qlik · tableau · T-Systems

Platinum

Cherwell Software · HIXSA · gmc Software · IMPERVA · mega Managing Enterprise Complexity · outsystems · redhat · salesforce · servicenow · Softtek · SOTI

Guía de Arranque para VR con Oculus y Leap Motion Orion

—

Por Ernesto Riestra

Desde sus inicios, la realidad virtual ha sido una gran promesa de interacción con los sistemas digitales. Actualmente esta promesa ya es una realidad gracias a plataformas y dispositivos como Oculus Rift y Leap Motion.

Para poder lograr una sensación de inmersión completa, la realidad virtual incorpora dos elementos clave en su parte visual: primeramente la estereoscopía, que significa proyectar dos imágenes diferentes para cada ojo, de manera que logremos la sensación de profundidad, el tracking de la posición y orientación de la cabeza.

El segundo elemento es la interacción. La reacción inmediata al entrar a un entorno de VR, es extender las manos y tratar de tocar o alcanzar las cosas que están en el mismo. Aunque existe la posibilidad de usar controladores físicos, como gamepads, joysticks y sistemas de tracking de posición 3D, no hay nada como la idea de usar las propias extremidades para interactuar. Justamente esta es la idea detrás de innovaciones como Kinect de Microsoft o Leap Motion, un sensor que detecta movimientos y gestos hechos con las manos, lo cual permite interacciones más libres e intuitivas.

En este recorrido vamos a explorar todos los pasos necesarios para desarrollar un entorno de VR basado en Oculus y Unity, soluciones disponibles y accesibles en México. También revisaremos algunos ejemplos con Orion, la versión más reciente de software para Leap Motion.

PRERREQUISITOS E INSTALACIÓN BÁSICA

Para poder realizar este tutorial se necesita lo siguiente:

- Computadora con requerimientos mínimos para Oculus y sistema operativo Windows
- Unity 5.3.5
- Oculus Runtime 1.3
- Leap Motion Orion Software

En este tutorial utilizamos una computadora Alienware X51 R2, con tarjeta gráfica AMD Radeon R9 370 de 4GB y sistema operativo Windows 10 Home.

Aunque Oculus nos señalaba que nuestro equipo era una computadora no soportada, no experimentamos ningún problema de rendimiento con la versión Developer Kit 2 (DK2). Lamentablemente Oculus dejó de soportar hardware de Apple, por lo que no es posible reproducir este tipo de experiencias en sus equipos, al menos con el soporte y versiones más recientes al momento de escribir el presente artículo.

Nota: Para configurar el hardware hay que seguir las guías de producto de Oculus y de Leap Motion respectivamente. La instalación de Unity puede ser la versión Personal Edition sin costo.

Esta ya contiene las librerías básicas de compatibilidad, así que es necesario descargar de la página de Oculus los drivers más recientes para evitar inconvenientes.

LEAP MOTION ORION EN UNITY

Una vez instalado todo el hardware necesario, es momento de configurar Unity para desarrollo con Leap Motion Orion. Aunque se trata de una versión Beta, tiene enormes mejoras en la reducción de ruido y detección de gestos con las manos.

1. Para configurarlo dirígete a la página de desarrollo de Leap Motion Orion. Visita la sección de Unity: https://developer.leapmotion.com/unity, y da click en Download Core Assets. Además podrás descargar todos los add-on modules; funciones que no son parte de los core assets, pero que contienen metáforas de interacción interesantes y son soluciones a la medida de frecuente uso. Todos estos elementos se presentan como .unitypackage, formato comprimido de Unity para almacenar activos que puedes integrar en un proyecto.

2. El segundo paso es abrir Unity y crear un proyecto nuevo (o usar un proyecto previo si se cuenta con él). Ya dentro del proyecto dirígete a la ruta: Asset/Import Package/Custom Package... para instalar los paquetes.

3. Ahora selecciona cada paquete; automáticamente se almacenarán assets, scripts, scenes y otros elementos en la librería (ver figura 1).

Ernesto Riestra es cofundador y líder de investigación y desarrollo en Metagraphos, firma dedicada a la creación de experiencias digitales para transformar la productividad y el conocimiento de personas y organizaciones. Es Ingeniero Mecánico Electricista egresado de la UNAM y tiene una Maestría en Mecatrónica por la Universidad Técnica de Hamburgo.

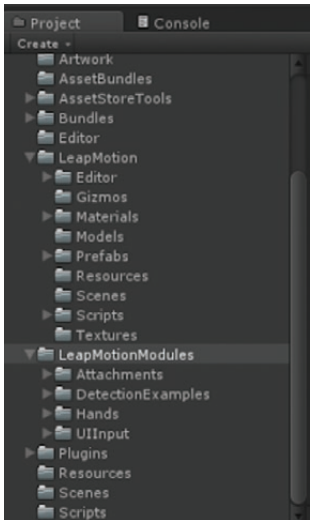


Figura 1. Árbol de carpetas.

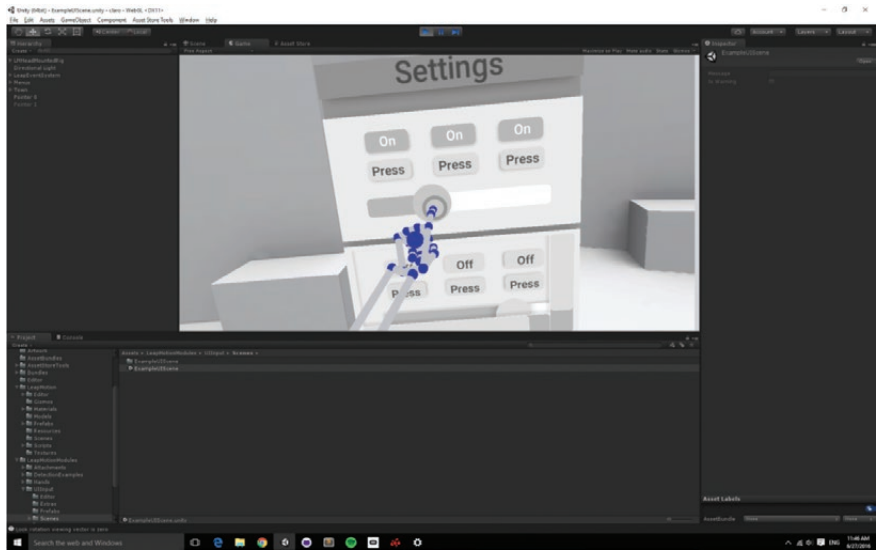


Figura 3. Interacción con componentes 2D.

Las escenas de ejemplo están disponibles en las carpetas LeapMotion y LeapMotionModules; puedes probar algunas de las más interesantes. Describo algunos a continuación:

Leap_Hands_Demo_VR. Encontrarás este demo bajo LeapMotion/Scenes. La escena muestra un entorno VR básico en el que se utiliza Oculus y Leap Motion para mostrar las manos del usuario. La sensación de inmersión, aun con un fondo en gris, es increíble. Además se puede constatar la precisión de seguimiento y registro, es decir, la coincidencia entre la imagen virtual y nuestra sensación de dónde están nuestras manos (ver figura 2).

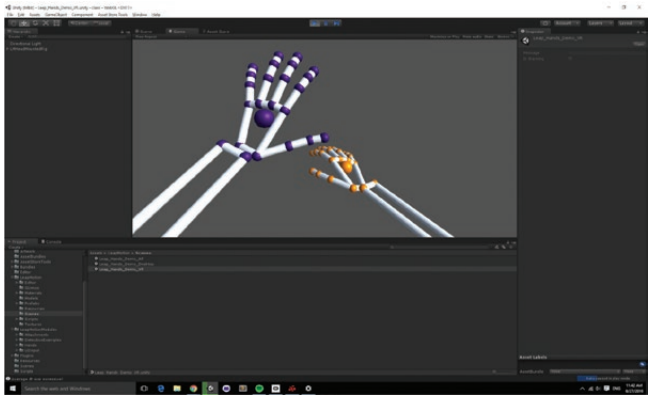


Figura 2. Demo de registro de manos.

ExampleUIScene. Esta escena se encuentra dentro de la carpeta *LeapMotionModules/UIInput/Scene*s y contiene ejemplos de interfaces de usuario 2D insertadas en un entorno de VR. La interacción es realmente fluida y los botones de ejemplo (tipo click, toggle y slider) reaccionan muy bien.

PinchDrawDemo. Es otro ejemplo muy interesante. Con un poco de práctica es posible escribir y dibujar en un espacio 3D (ver figura 4). La encontrarás en LeapMotionModules/DetectionExamples/Scenes.

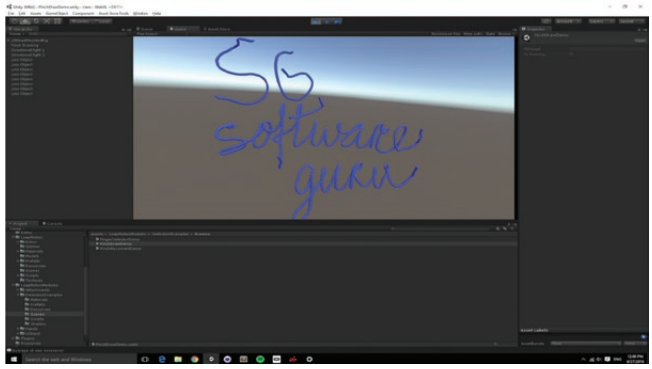


Figura 4. PinchDrawDemo.

INTEGRANDO LEAP MOTION A TU ESCENA

Para integrar Leap Motion a una escena hay que crear algunos objetos desde la carpeta de prefabs de Leap Motion. Estos objetos contienen una cámara configurada para ser el punto de vista en VR y que el visor de Oculus lo pueda controlar, también están disponibles los objetos que representan las manos derecha e izquierda.

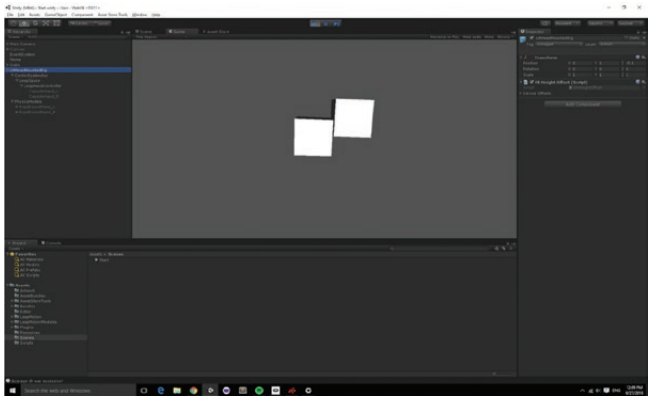


Figura 5. Escena básica.

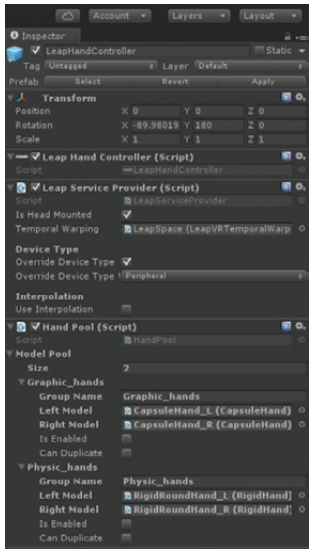


Figura 6. Configuración de HandPool.

La escena utilizada para este tutorial es muy básica, un par de cubos animados que se desplazan relativamente (ver figura 5).

Para realizar dicho recorrido:

1. Ir a LeapMotion/Prefabs en la ventana de assets y arrastrar el prefab LMHeadMountedRig a la escena.

2. En la jerarquía de LMHeadMountedRig, hay un objeto llamado LeapHandController.

3. Desde el folder LeapMotion/Prefabs/HandModelsNonhuman, arrastrar los prefabs CapsuleHand_L y CapsuleHand_R a la ventana de la jerarquía y meter bajo LeapHandController.

4. Desde el folder LeapMotion/Prefabs/HandModelsPhysical, arrastrar el prefab RigidRoundHand_L y RigidRoundHand_R a la ventana de la jerarquía y meterlos igualmente bajo LeapHandController.

5. En la ventana del inspector, ubicar el objeto HandPool que está dentro de LeapHandController.

6. Debes crear un array en Model Pool; para hacerlo, escribe en Size el valor 2.

7. Arrastrar cuatro objetos en la escena (manos derecha e izquierda respectivamente), para que el controlador los pueda referir. Como se muestra a continuación, esto corresponde a dos manos gráficas y dos manos físicas.

8. Una vez creado este array de dos elementos, marcar los checkboxes IsEnabled y CanDuplicate de cada set de manos.

9. Hay que asegurarse de que esté habilitado el soporte a VR; ubicar el checkbox en Edit/Project Settings/PlayerSettings/OtherSettings/Virtual Reality Supported.

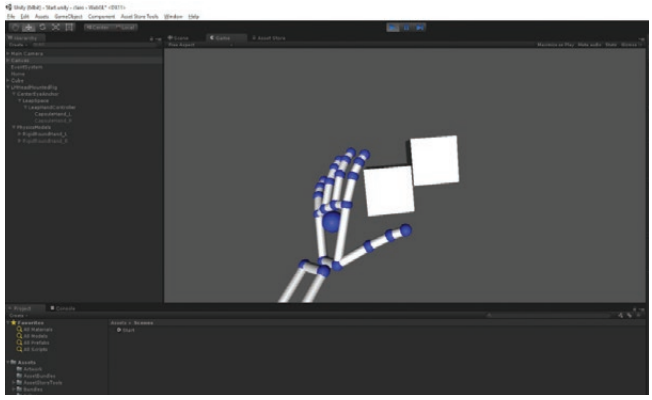


Figura 7. Interacción con objetos rígidos.

10. Para la interacción con objetos es importante que se trate de objetos rígidos (agregando un componente rigidBody a los mismos).

CONCLUSIONES

Con estos sencillos pasos, puedes crear tu propio entorno en VR. Además tienes la oportunidad de verificar el gran avance que hay en cuanto a estas dos tecnologías; combinadas permiten realizar grandes cosas como escenas VR con interacción a través de las manos y no de dispositivos de apoyo.

Lo mejor es que cualquiera puede revisar y analizar estas herramientas y escenas demo, para integrarlas en sus propios desarrollos. Espero que este tutorial haya sido de tu interés, y que te anime a desarrollar tus propias ideas en un entorno de realidad virtual.

SG® talento

Donde se encuentran los prefesionistas que crean software grandioso

BENEFICIOS DE TENER TU PERFIL EN SG TALENTO:

- Muestra tu experiencia y habilidades en tu perfil público
- Contribuye a generar información estadística sobre el talento de TI en nuestra región.
- Si estas buscando oportunidades laborales, podrás contactar con reclutadores y aplicar a las vacantes disponibles.
- Además podrás participar en promociones especiales organizadas por SG: descuentos, rifas, concursos, entre otros.

www.sgtalento.com

Desarrollando un Asistente Personal Inteligente

QUIERO MI PROPIO SIRI

—

Por Pedro Ramírez

"Siri, lee los mensajes de correo electrónico"
"Hola Cortana, ¿cómo está el clima?"
" Ok Google, ¿cómo se dice hola en Inglés?"
"Alexa, set a timer for 15 minutes"

Este tipo de frases (comandos) se están haciendo parte de nuestra vida cotidiana. Cada día es más común usar nuestra voz y un lenguaje natural para interactuar con dispositivos y servicios. La gran mayoría de los dispositivos y sistemas operativos más populares incorporan lo que se conoce como un asistente personal inteligente; es una tendencia que se está acelerando.

Un asistente personal inteligente (Intelligent Personal Assistant, IPA) es un agente de software que puede realizar tareas u ofrecer servicios a un individuo. El usuario puede interactuar con el IPA usando lenguaje natural a través de voz o texto. El IPA puede además obtener y usar información adicional de contexto tal como la ubicación del usuario, información dentro del mismo dispositivo (fotos, contactos, etc.) y otros servicios para proporcionar una mejor respuesta al usuario.

Los asistentes personales como Siri, Google Now o Cortana pueden ejecutar muchas acciones y responder a una gran variedad de peticiones del usuario. Típicamente, el dispositivo es tan solo una fachada para proveer el servicio, ya que en realidad los servicios que reconocen el habla y determinan la respuesta o acción adecuada, son servicios hospedados en la nube.

QUIERO MI PROPIO IPA

Si queremos crear un servicio de IPA, una opción es hacerlo como una extensión a servicios existentes como Siri, Cortana o Alexa. Sin embargo, esto puede tener limitaciones o no darnos el control completo que podríamos requerir. Así que otra opción es que desarrollemos nuestra propia aplicación y simplemente utilicemos los servicios de reconocimiento del habla que exponen algunos de estos motores.

Para construir nuestro IPA primero necesitamos identificar sus distintos componentes o subsistemas. A grandes rasgos son: interfaz para interactuar con el asistente, reconocimiento del habla, lenguaje de dominio específico (DSL), y endpoint para acceder al agente.

A continuación vamos a platicar sobre cada uno de ellos.

RECONOCIMIENTO DE HABLA

Es un campo de la lingüística computacional orientado a desarrollar metodologías y tecnologías que permitan que las computadoras puedan entender y traducir lenguaje hablado. Vale la pena aclarar que el reconocimiento del habla es distinto del reconocimiento de voz, ya que este último solo se enfoca en identificar a la persona que habla, mas no lo que está diciendo.

Como pueden imaginar, desarrollar un servicio de reconocimiento del habla es una tarea muy grande y casi imposible para una sola persona. Afortunadamente empresas como Microsoft, Google y Xamarin entre otros, los han desarrollado ya, y los han abierto para que podamos usarlos en nuestras aplicaciones. En el listado 1 muestro cómo podemos inicializar el servicio de reconocimiento del habla en .Net. Posteriormente, en el listado 2 muestro cómo se podría hacer en web utilizando el Web Speech API, que es soportado por Chrome.

```
var Language = (from language in InstalledSpeechRecognizers.All
                where language.Language == "es-ES"
                select language).FirstOrDefault();

SpeechRecognizerUI speechRecognition = new SpeechRecognizerUI();
speechRecognition.Recognizer.SetRecognizer(Language);
SpeechRecognitionUIResult recoResult = await speechRecognition.RecognizeWithUIAsync();

if (recoResult.ResultStatus == SpeechRecognitionUIStatus.Succeeded) {
    txtPregunta.Text = recoResult.RecognitionResult.Text.Replace(".","");
    LaunchSearch();
}
```

Listado 1. Código C# para usar Windows Speech Recognition.

Pedro Ramírez Suárez es Chief Architect en Scio Consulting, desarrollador independiente de juegos, siempre aprendiendo y buscando nuevas formas de usar la tecnología.
https://github.com/pedro-ramirez-suarez

```
recognition = new webkitSpeechRecognition();
recognition.lang = "es-MX";
recognition.continuous = true;
recognition.interimResults = true;
recognition.onresult = function (event) {
    var interim_transcript = '';
    for (var i = event.resultIndex; i < event.results.length; ++i) {
        if (event.results[i].isFinal) {
            final_transcript += event.results[i][0].transcript;
        } else {
            if (event.results[i][0].transcript != '')
                interim_transcript += event.results[i][0].transcript;
        }
    }
    if (interim_transcript != '') {
        $('#query').val(interim_transcript);
        if ($.trim(final_transcript) != $.trim(interim_transcript))
            final_transcript = interim_transcript;
    }
}
recognition.start();
```

Listado 2. Código Javascript para usar Web Speech API.

LENGUAJE DE DOMINIO ESPECÍFICO (DSL)

Es un lenguaje informático diseñado para representar o resolver un problema específico. Existe una gran variedad de DSLs, desde lenguajes de uso muy común como HTML o SQL, hasta lenguajes de nicho como podría ser CSound para síntesis de sonidos. El dominio puede ser también un área de negocios. Así que por ejemplo, podríamos crear un DSL para gestión de pólizas de seguro.

Cuando se crea un DSL que solo se usará en una aplicación se le acostumbra llamar un mini-lenguaje.

Para implementar el lenguaje necesitamos definir su gramática y crear un parser (analizador sintáctico) para poder interpretar lo que el usuario nos dice y darle una respuesta. Podríamos hacer las dos cosas a mano, pero en la actualidad existen ya muchas herramientas que nos facilitan definir la gramática y crear nuestro parser.

Una de las herramientas más comunes para desarrollar lenguajes es YACC, que nos permite generar un parser basado en una gramática analítica que debemos describir en un metalenguaje similar a la notación de Backus-Naur.

En este caso no usaremos YACC, sino que usaremos Irony, un kit de desarrollo para implementar lenguajes en la plataforma .Net. A diferencia de soluciones como YACC y Lex, que generan un parser a partir de la gramática analítica que hayamos definido, Irony nos permite expresar la gramática de nuestro lenguaje directamente en C#.

El listado 3 muestra cómo definimos la gramática a través de terminales y no-terminales, así como los enunciados que vamos a reconocer.

```
//Terminals
  var cuantos = ToTerm("cuantos");
  var cuantas = ToTerm("cuantas");
  var cual = ToTerm("cual");
  var que = ToTerm("que");
  var cuando = ToTerm("cuando");
  var tiene  = ToTerm("tiene");
  /* más aquí */
  var number = new NumberLiteral("number");
  var number2 = new NumberLiteral("number2");
  /* más aquí */

  //non terminals
  var cuantosStatement = new NonTerminal("cuantosStatement");
  var queStatement = new NonTerminal("queStatement");
  var actualizaStatement = new NonTerminal("actualizaStatement");
  var operacionesStatement = new NonTerminal("operacionesStatement");
  /* más aquí */

  //Cuanto pregunta
  cuantosStatement.Rule = (cuantoscuantas + tabla + tiene + tabla2 + whereId) |
                          (en + cuantoscuantas + tabla + esta + tabla2 + whereId);
  //que pregunta
  queStatement.Rule = (que + campo + tiene + tabla + whereId) |
                      (que + tabla + campo + antesdespues + de +whereId );
  /* más aquí */
  //actualiza comando
  actualizaStatement.Rule = actualiza + ella + tabla + whereCampo + cambia + su + campo + por + valueSet;
  //operaciones matematicas
  operacionesStatement.Rule = cuanto + es + number + (multiplicacion | suma | resta | division) + number2;

  //todos los comandos posibles de nuestro lenguaje
  comando.Rule = cuantosStatement | queStatement | cuandoStatement | cualMasMenosStatement |
                 cualMayorMenorStatement | muestraTodoStatement | muestraUnoStatement |
                 actualizaStatement | operacionesStatement;
```

Listado 3. Definición del DSL.

ENDPOINT

Vamos a exponer los servicios de nuestro asistente a través de un endpoint alojado en nuestra aplicación web. Este endpoint recibirá como parámetro la consulta del usuario y la pasará al asistente, el cual interpretará los resultados y enviará una respuesta serializada.

```
var client = new RestClient(AppResources.DeepThoughtRoot);
client.AddHandler("application/json", new DynamicSerializer());

client.ExecuteAsync<dynamic>(new RestRequest(AppResources.DeepThoughtSearch + txtPregunta.Text), (res) => {
  var data = JsonConvert.DeserializeObject<dynamic>(res.Content);
  SpeechSynthesizer synth = new SpeechSynthesizer();
  VoiceInformation spvoice;
  bool found = true;
  // show results
  foreach(var e in data) {
    string result = e.ToString();
```

Listado 4a. Disparar búsqueda y procesar resultados.

```
    var record = result.Split(new char[] { '|'});
    var row = new StackPanel { Background = new SolidColorBrush(Color.FromArgb(255, 40,40,40)),
                               Margin = new Thickness(2,2,2,8) };

    foreach (var f in record) {
      if (string.IsNullOrWhiteSpace(f))
        continue;
      var sp = new StackPanel { Orientation = System.Windows.Controls.Orientation.Horizontal,
                                Margin = new Thickness(2, 2, 2, 8) };
      var item = f.Split(new char[] { ':' });
      if (!string.IsNullOrWhiteSpace(item[0])) {
        var field = new TextBlock { Text = item[0], Foreground = new SolidColorBrush(Color.FromArgb(255, 30, 144, 255)),
                                    FontSize = 30 };
        sp.Children.Add(field);
      }
      var value = new TextBlock { Text = item[1], Foreground = new SolidColorBrush(Color.FromArgb(255, 211, 211, 211)),
                                  FontSize = 30 };
      sp.Children.Add(value);
      row.Children.Add(sp);
    }
    results.Children.Add(row);
  }
```

Listado 4b. Desplegar resultados.

```
  //set voice and message
  string msg = string.Empty;
  if (found) {
    msg = "Esto es lo que encontré";
    spvoice = InstalledVoices.All
              .Where(voice => voice.Language.Equals("es-ES") & voice.Gender == VoiceGender.Female)
              .FirstOrDefault();
  }
  else {
    msg = "Lo siento, no puedo entenderte, intenta de nuevo.";
    spvoice = InstalledVoices.All
              .Where(voice => voice.Language.Equals("es-ES") & voice.Gender == VoiceGender.Male)
              .FirstOrDefault();
  }
  results.Children.Add(new StackPanel { Height= 60 });
  synth.SetVoice(spvoice);
  synth.SpeakTextAsync(msg);
```

Listado 4c. Dar mensaje hablado.

APP

El usuario podrá interactuar con nuestro asistente inteligente a través de una aplicación móvil.

Ya en los listados 1 y 2 vimos cómo podemos usar las APIs de Chrome y Windows para reconocimiento del habla. Ahora en los listados 4a, 4b y 4c podemos ver cómo lanzar la búsqueda y mostrar los resultados en una aplicación Windows Universal.

El listado 4a muestra cómo disparar una petición asíncrona y desmenuzar los resultados, luego el listado 4b muestra cómo se podrían desplegar y por último el listado 4c muestra cómo podríamos acompañar estos resultados de un mensaje de voz.

ALTERNATIVAS

Un asistente personal inteligente puede desarrollarse de muchas maneras. En este artículo tomamos algunas decisiones en cuanto a tecnologías y arquitectura, para la parte de reconocimiento del habla podríamos usar servicios de Xamarin que tiene una API de reconocimiento de lenguaje y es multiplataforma. También podríamos integrar nuestra aplicación con servicios como Siri o Cortana. El DSL, como mencionamos, también podríamos haberlo desarrollado con YACC, Lex o alguna variante de estos.

CÓDIGO FUENTE

Todo el código fuente en este artículo es parte de un asistente personal pequeño que está hospedado en http://deepthoughtagent.azurewebsites.net/ , en la url: http://deepthoughtagent.azurewebsites.net/Home/preguntas están el tipo de preguntas que puede contestar el asistente; algunos ejemplos son:

- ¿Cuántas embarcaciones tiene contrato a123456?
- ¿En cuántos contratos esta embarcación uno?
- ¿Qué capacidad tiene embarcación Morelos?
- ¿Qué contrato empieza antes de '12/12/2015'?
- ¿Cuál embarcación tiene mayor velocidad?
- ¿Cuál contrato tiene menos embarcaciones?
- ¿Cuándo empieza contrato ABCDE?
- Muestra todas las embarcaciones
- Muestra todos los contratos
- Muestra el contrato ABCDE
- Actualiza la embarcación pinta cambia su velocidad por 20
- ¿Cuánto es 2 + 2?
- ¿Cuánto es 2 entre 2?

El código fuente se puede obtener en https://github.com/pedro-ramirez-suarez/ScioAssistant, la aplicación móvil (Windows Universal App) está configurada para usar el endpoint http://deepthoughtagent.azurewebsites.net/, siendo que el servicio está hospedado en un servidor con recursos muy limitados no va a poder manejar muchas peticiones, para correr sus propios experimentos basta con hospedar el DSL en otro lugar y cambiar la URL definida en el archivo "AppResources.resx" para que apunte a donde está nuestro endpoint.

5 Pasos Estratégicos para Crear una Startup a Partir de una Idea

Por Edith Gomez

Muchas personas cuentan con ideas fabulosas, pero no consiguen llevarlas a su fin. Si te encuentras cansado de que tus ideas solo sean eso, empieza a tomar acción y a poner las cartas sobre la mesa. ¿No tienes claro por dónde comenzar?

Te hemos preparado una guía personalizada para emprendedores, con claves y una ruta a seguir para que tu idea deje de serlo y se convierta en una gran empresa. Todo se puede conseguir, siempre que seas constante.

1. DEFINIR LA IDEA

Intenta ser lo más específico posible. Ningún negocio, ni pequeño ni grande, puede considerar que tenga todo para todas las personas del mundo. Ten en mente que cuanto más específica sea la idea más fácil será centrarse en un nicho de mercado y encontrar al público ideal. De todas formas, si no es posible, deberás restringir al máximo la competencia, aportando servicios extra, nuevos productos, etc. Esa es la clave para la mayoría de las empresas que hay en el mercado.

Considera todas las facetas de la idea. Convertir una idea en un negocio requiere que pensamos acerca del producto o servicio que vamos a ofrecer. Debemos contextualizarlo y averiguar cómo encajaría en el mercado.

Partiendo de esta base, piensa que al crear y definir tu idea como emprendedor, haz de plantearte las siguientes preguntas:

1. ¿Qué es lo que vendo específicamente?
2. ¿Qué funciones realiza el producto que vendo?
3. ¿En qué se diferencia o es mejor de otros productos o servicios que ya existen?
4. ¿Quién va a ser mi público objetivo?
5. ¿Por qué comprarían mi producto o servicio y no otro?
6. ¿Cómo puedo promocionar mi producto o servicio?
7. ¿Quién es la competencia?

Ten claro que cuando la idea comience a tomar forma, teniendo en cuenta su papel en las diferentes facetas del proceso de conversión a empresa, es el momento ideal de cuándo comenzamos a poder evaluar con mayor precisión los riesgos potenciales y los escollos a los que nos vamos a enfrentar.

2. ENCAJAR LA IDEA EN EL MERCADO

Llevar a cabo un estudio de mercado te permite evaluar si tu idea vale la pena para invertir en ella, o no. Comienza la búsqueda escribiendo cuál crees que es la principal necesidad o problema, el cual puede solucionar tu producto o servicio. Escríbela y mantén ese papel cerca de ti, bien presente en todo momento.

Descubre cuánta gente tiene ese problema o necesidad a la que vas a dar una solución y selecciona a algunas de estas personas para hablar sobre ello que a todos interesa. Considera crear una encuesta para realizarla a dicho público objetivo, con el fin de averiguar qué opinan sobre tu idea y qué alternativas de solución les gustaría para su necesidad o problema. Así sabrás de primera mano qué quieren tus posibles clientes y cómo adaptar tu idea de negocio a sus necesidades.

Cuando ya cuentes con unos buenos resultados, investiga a la competencia y averigua si tu proyecto es diferente al de ellos y si realmente puede competir contando con muchas opciones de ganar.

3. DEFINIR LA MARCA

La marca que elijas será tu identidad, conformando el primer paso a la hora de empezar una empresa. Esto debe ser antes de comenzar con la creación de la empresa en términos legales, porque en este punto decidirás el nombre que debe tener, la definirás.

A medida que pase el tiempo podrás detallar todo mucho más, pero debes saber cómo va a ser tu marca con el fin de orientar tu estrategia de negocio de forma clara y definida.

Cosas a tener en cuenta a la hora de escoger un nombre:

- Elige un nombre que describa el problema que vas a tratar.
- Asegúrate de averiguar el éxito y el fracaso asociado a ese nombre y pregunta a tus posibles clientes qué les parece.

Es el momento perfecto de asegurarse un dominio web, con el fin de que tu nombre sea tuyo por fin y de nadie más en todo internet.

4. REGISTRAR LA EMPRESA Y ASEGURAR LA PROPIEDAD INTELECTUAL

Si ya tienes claros los pasos anteriores – la idea y cómo puede funcionar – el siguiente paso se centra en patentarla y asegurarte de que nadie más pueda desarrollar de igual forma (aunque ten claro que siempre aparecerán imitadores). La propiedad intelectual se centra en que un individuo o compañía posea los derechos de un producto o empresa determinada.

Es fundamental que sigas las pautas legales para patentar tu negocio y comenzar el proceso de su creación fielmente. Para este paso te aconsejamos que recopiles toda la información necesaria de las páginas oficiales del estado, con el fin de utilizarlas como guía en tu proceso. Anota todas las preguntas que te surjan y dirígete a una de las oficinas más cercanas para solucionar tus cuestiones.

5. CONSEGUIR CAPITAL

Haz un plan de negocio. Este puede ser el paso más complicado y también el que más se distancie en el tiempo. Debes crear un plan de negocio que cuente con las partes necesarias para potenciar una startup nueva e innovadora. Por lo tanto, deberás crear campañas de venta, estrategias de marketing, definir el branding a fondo, etc. Es el momento perfecto para entrar en materia y convertir los primeros planes en realidad.

Conseguir dinero y comenzar a crear la startup son cosas que pueden ir unidas. Además, considera que si consigues un ángel en el negocio, te aportará apoyo financiero pero también consejos y gestión sobre la dirección. El proceso de creación de tu negocio se verá muy influido por la forma en la que consigas financiarlo. Si vas a depender de inversores y terceras personas, tendrás que mantenerlos al tanto de tus movimientos, y aunque el banco te dé un préstamo o capital, deberás tener en mente devolverlos con un porcentaje extra de intereses.

Referencias

[1] M. Parreño, "5 formas de financiamiento para emprendedores sin dinero", Gananci. http://swgu.ru/ro

Edith Gómez (@edigomben) es editora en gananci.com, apasionada del marketing digital, especializada en comunicación online. Se niega a irse a la cama cada noche sin haber aprendido algo nuevo. Le inquietan las ideas de negocio y, más aún, aportar una mirada creativa al pequeño mundo en el que vivimos.

ARQUITECTURAS
DE SOFTWARE PARA LA NUBE

Hoy en día, tener aplicaciones en la nube no es ninguna novedad, es el estándar de facto. Aunque el concepto de una "nube de cómputo" existe desde el siglo pasado, la noción que tenemos actualmente del cómputo en la nube se comenzó a hacer popular a partir del año 2006, cuando Amazon lanzó su servicio Elastic Compute Cloud. Desde entonces, en la industria hemos discutido ampliamente sobre todo tipo de aspectos del cómputo en la nube: sus ventajas, consideraciones de seguridad, distintas modalidades (SaaS, IaaS, PaaS), etcétera. Pero uno de los aspectos al que se le ha dado poca atención es al del impacto que la nube tiene en la forma en la que estructuramos nuestras aplicaciones.

En principio, podemos argumentar que una aplicación cliente-servidor sigue teniendo la misma estructura independientemente de si el servidor es una computadora en la red local o un servidor virtualizado y replicado a través de distintos centros de datos distribuidos geográficamente. Esta postura es cierta, pero algo miope, ya que desaprovecha las capacidades naturales del cómputo en la nube, tales como la elasticidad y el modelo de pago por uso. Si queremos aprovechar las ventajas de la nube necesitamos diseñar nuestras aplicaciones de una forma distinta, una forma que promueva la escalabilidad horizontal y tenga fronteras claras de descomposición.

Otra tendencia que ha impactado fuertemente nuestra industria en los últimos años es la del procesamiento masivo de datos (big data). Lo que también afecta de forma significativa la forma en la que diseñamos arquitecturas para el procesamiento de datos. Así que es importante que estemos al tanto de mejores prácticas arquitectónicas para atender necesidades de procesamiento intensivo de datos.

En las siguientes páginas compartimos varios artículos que abordan algunas de las principales tendencias arquitectónicas relacionadas con la nube y el big data. Esperamos que ayuden a traer claridad sobre este tema.

APLICACIONES DE 12 FACTORES

LINEAMIENTOS PARA CONSTRUIR APLICACIONES NATIVAS EN LA NUBE

Por Pedro Galván

Si estás involucrado en la arquitectura de aplicaciones de software que se ejecutan en contextos de cómputo en la nube, posiblemente estés familiarizado con el término *twelve-factor application* (aplicación de 12 factores); si no, deberías de estarlo.

Este es un concepto utilizado para describir aquellas aplicaciones que han sido diseñadas y programadas para operar de forma adecuada en la nube. Para lograrlo, deben cumplir 12 requisitos o factores. En este artículo los explico y contextualizo.

¿DE DÓNDE VIENEN LOS 12 FACTORES?

Como probablemente ya has leído en muchos lados, incluyendo las páginas de SG, hay tres modalidades principales de cómputo en la nube: SaaS (software como servicio), IaaS (infraestructura como servicio) y PaaS (plataforma como servicio). La promesa de la modalidad PaaS es que una vez que has construido tu aplicación, la subes a un PaaS y tu aplicación automáticamente opera desde la nube sin necesidad de que tú tengas que preocuparte por cuestiones de infraestructura tales como mantener el sistema operativo, almacenamiento, configuración de puertos, etcétera. Tú simplemente indicas qué frameworks y servicios requiere tu app, la subes, y la infraestructura se genera y gestiona de forma automática.

El modelo PaaS suena maravilloso, pero para sacarle provecho se requiere que las aplicaciones desarrolladas sean "amigables con la nube". Una aplicación amigable con la nube no es aquella que simplemente puede ejecutarse en la nube; sino que también escala de forma elástica, no tiene estado (stateless), usa filesystems efímeros, y trata todo como un servicio. Las aplicaciones construidas de esta manera se pueden desplegar y escalar rápidamente.

El problema es que es un modelo nuevo con el que la mayoría de los desarrolladores no tiene experiencia previa, por lo que desconoce las consideraciones arquitectónicas que debe tener al construir aplicaciones de este tipo.

Para resolverlo, un grupo de personas en la empresa Heroku (uno de los proveedores pioneros de PaaS) definió los 12 factores. En esencia, son un manifiesto que describe las reglas y lineamientos que deben seguirse para construir una aplicación nativa a la nube. El objetivo es mostrar cómo construir aplicaciones que estén desacopladas del sistema operativo, se puedan desplegar de forma automatizada, y escalen de forma dinámica.

A continuación describo los factores.

I. BASE DE CÓDIGO

Una base de código registrada en control de versiones, muchos despliegues.

Cada aplicación debe tener un (y solo un) repositorio de código. El repositorio debe tener control de versiones, y a partir del mismo repositorio se generan las instancias de la aplicación. Para instanciar distintas versiones de una aplicación (ej. producción, staging, desarrollo) se usa el mismo repositorio de código, pero con ramas distintas.

II. DEPENDENCIAS

Declarar dependencias explícitamente y aislarlas.

La aplicación declara sus dependencias —bibliotecas, componentes externos— de forma explícita y precisa por medio de un manifiesto o especificación. Adicionalmente, se debe utilizar una herramienta para aislar dependencias y de esta forma evitar que se "cuelen" dependencias implícitas provenientes del ambiente de ejecución.

Por ejemplo, en Ruby se puede usar Gem Bundler para declarar dependencias y bundle exec para aislar dependencias. En Python se usa Pip para declaración y Virtualenv para aislamiento. Incluso en C se puede usar Autoconf para declaración y enlaces estáticos (static linking) para aislamiento. No importa cuáles sean las herramientas, la declaración de dependencias siempre se debe usar en conjunto con un mecanismo de aislamiento.

Las aplicaciones tampoco deben depender de herramientas de sistema. Así que una aplicación que invoque por medio de línea de comando cierta herramienta (ej. curl o ImageMagick) está violando este factor. Lo que se debe hacer es incorporar dicha herramienta como una biblioteca y especificarla como dependencia.

BIO. **Pedro Galván Kondo** es cofundador y director editorial de Software Guru.

III. CONFIGURACIÓN

Guardar la configuración en el entorno.
La configuración de una aplicación está relacionada con aquellos parámetros que pueden variar dependiendo del ambiente (producción, pruebas, desarrollo). Esto incluye: urls a otros servicios, contraseñas de base de datos, valores que varían en cada instancia.

Guardar estos valores como constantes en el código es una práctica no segura y que viola los 12 factores. La configuración puede variar entre instancias, pero el código no debe variar. La práctica recomendada consiste en que la configuración se haga disponible a la aplicación por medio de variables de entorno (env vars).

IV. SERVICIOS EXTERNOS

Tratar a los servicios externos como recursos conectables.
Un servicio externo es cualquier servicio que la app consume por medio de la red como parte de su operación normal. Esto incluye bases de datos, sistemas de mensajería, servidores SMTP para enviar correos, entre otros. Estos servicios algunas veces son locales y operados por los mismos administradores que la aplicación, pero también pueden ser servicios remotos proporcionados por terceros.

La aplicación no debe hacer distinción entre servicios locales y de terceros. Ambos son recursos conectados, accesibles por medio de un url u otras credenciales guardadas en la configuración. Esto permite que sea posible reemplazar el uso de una base de datos local en MySQL por una base de datos remota, como Amazon RDS, cambiando solamente la configuración y sin necesidad de hacer ninguna modificación en el código.

V. CONSTRUIR, LIBERAR, EJECUTAR

Separar claramente las fases de construcción y ejecución.
Una base de código se transforma en un despliegue (*deploy*) por medio de tres fases:

- En la fase de construcción (*build*) se genera un paquete ejecutable a partir de la base de código. El paquete ejecutable puede contener dependencias (bibliotecas), código ejecutable (ya sea binario o interpretado) y activos (ej. Imágenes, hojas de estilo).

- La fase de liberación (*release*) toma el paquete ejecutable y lo combina con una configuración de despliegue. El paquete resultante contiene tanto el build como la configuración y está listo para ejecutarse.

- En la fase de ejecución (*runtime*) se opera la aplicación por medio de la ejecución de un conjunto de procesos de una liberación específica de la aplicación.

Separar estas fases involucra que no se pueden realizar acciones que no correspondan a dicha fase. Por ejemplo, no se puede hacer cambios al código durante la fase de ejecución ya que no hay forma de propagar esos cambios a la fase de construcción.

VI. PROCESOS

La aplicación se ejecuta en forma de procesos sin estado.
En el escenario más simple, una aplicación consiste de un solo script y el ambiente de ejecución es una computadora personal con el runtime, y el proceso es lanzado por medio de línea de comandos (por ejemplo, `python mi_script.py`). asdf

La aplicación opera como procesos independientes que no comparten nada entre sí. Cualquier dato que requiera guardarse debe ser almacenado en un servicio externo de persistencia, típicamente una base de datos. El espacio en memoria o sistema de archivos se puede usar como caché temporal para transacciones individuales, pero nada más. La aplicación nunca puede asumir que algo cacheado en memoria o disco estará disponible para una petición futura, ya que no hay ninguna garantía de que una petición futura sea atendida por el mismo proceso. Incluso cuando la aplicación se ejecute en un proceso único en un solo nodo de cómputo, este puede ser reiniciado en cualquier momento. Algunas aplicaciones web utilizan "sesiones pegajosas" (*sticky sessions*) —es decir, almacenar el estado de la sesión del usuario en memoria del proceso y esperar que las peticiones futuras sean ruteadas al mismo proceso. Esto es una violación de los 12 factores. Para guardar el estado de la sesión del usuario se puede usar un almacén con expiración de tiempo, como Redis o Memcached.

VII. ASIGNACIÓN DE PUERTOS

Publicar servicios mediante asignación de puertos.
Tradicionalmente, las aplicaciones web se ejecutan en el contexto de un contenedor web. Por ejemplo, Tomcat en el caso de aplicaciones Java, o Apache httpd + mod-php en el caso de PHP.

En contraste, una aplicación twelve-factor está completamente autocontenida y no depende de un servidor web en el entorno de ejecución. La aplicación exporta HTTP como un servicio que se asigna a un puerto y directamente atiende peticiones que llegan a dicho puerto.

Esto típicamente se implementa declarando una dependencia a una biblioteca de servidor web, tal como Tornado en Python, Thin en Ruby o Jetty en Java/JVM. Es así que todo es parte de la aplicación y solamente se expone el puerto correspondiente para atender peticiones.

Es posible exponer otros servicios además de HTTP por medio de asignación de puertos. Por ejemplo, una aplicación puede exponer un servicio de Redis o de XMPP, y a su vez esa aplicación puede operar como servicio externo (recurso conectable) para otra aplicación.

VIII. CONCURRENCIA

Escalar usando procesos.
Cualquier programa computacional, una vez en ejecución, es representado por uno o más procesos. Sin embargo, tradicionalmente las aplicaciones web tienen muy poca visibilidad hacia los procesos que la ejecutan.

En una aplicación twelve-factor los procesos son ciudadanos de primer nivel. Una aplicación puede definir distintos tipos de

proceso para manejar distintos tipos de petición Por ejemplo, las peticiones HTTP pueden manejarse por un proceso "web" mientras que las tareas de larga duración pueden ser manejadas por otro proceso "worker" que tiene diferente prioridad. Una aplicación puede tener varias instancias de un mismo tipo de proceso ejecutándose en el mismo nodo (escalar verticalmente) o puede agregar más nodos de cómputo y ejecutar los procesos en ellos (escalar horizontalmente).

Es importante que los procesos de la aplicación se puedan administrar por medio de los mecanismos provistos por el sistema operativo para administrar procesos (por ejemplo Upstart en Ubuntu).

IX. DESECHABILIDAD

Hacer el sistema más robusto por medio de arranque rápido y apagado gentil.
Los procesos de la aplicación son desechables, es decir que pueden ser iniciados y terminados en cualquier momento. Esto facilita la escalabilidad y el despliegue de cambios en el código o la configuración.

"Apagado gentil" es la traducción más cercana que se nos ocurre de *graceful shutdown*. En este caso se refiere a que cuando un proceso recibe una señal de finalización (SIGTERM) por parte del administrador de procesos debe: 1) dejar de escuchar el puerto de servicio, 2) finalizar las peticiones pendientes (o regresarlas a una cola de trabajo) y 3) terminar su ejecución.

X. PARIDAD ENTRE AMBIENTES

Mantener los ambientes de desarrollo, preproducción y producción lo más similares posible.
Tradicionalmente ha habido una brecha entre los ambientes de desarrollo y producción que se manifiesta en tres áreas:

- Brecha de tiempo: dado que las liberaciones a producción no son frecuentes, un desarrollador puede estar trabajando en código que tardará semanas o meses en estar en producción.

- Brecha de personal: el equipo de personas que construye la aplicación es distinto al que la pone en producción.

- Brecha de herramientas: en el ambiente de desarrollo se utiliza software distinto (herramientas, servidores, sistema operativo) que en producción.

En contraste, una aplicación twelve-factor está diseñada para entrega continua y para lograrlo reduce la brecha entre desarrollo y producción. Teniendo en cuenta las tres brechas descritas anteriormente, las acciones a tomar son:

- Reducir la brecha de tiempo entre liberaciones, haciendo varias liberaciones a la semana o incluso al día.

- Reducir la brecha en personal involucrando a los desarrolladores en el despliegue y monitoreo de la aplicación en producción.

- Reducir la brecha en herramientas manteniendo los ambientes de desarrollo y producción lo más parecidos posible.

XI. BITÁCORAS

Trata las bitácoras como corrientes de eventos.
Las bitácoras (*logs*) nos permiten ver cómo se está comportando una aplicación en ejecución. Típicamente, éstas se escriben en archivos en disco (*logfile*), pero es tan solo un formato de salida.

Una aplicación twelve-factor no se preocupa por rutear o almacenar su bitácora ni por gestionar los archivos de bitácora. En lugar de ello, cada proceso envía su corriente de eventos directamente a stdout. Es así que en desarrollo, el programador típicamente verá la corriente al ejecutar la aplicación desde la terminal de comandos. En ambientes de producción y preproducción, el de ejecución captura y organiza la corriente de eventos de cada proceso, y la rutea a uno o más destinos para que se archive. Estos destinos no son visibles ni configurables por la aplicación, son completamente administrados por el ambiente de ejecución.

Esto facilita que las bitácoras puedan ser indexadas y analizadas por medio de herramientas de propósito específico como Splunk, o incluso por sistemas más genéricos como Hadoop/Hive.

XII. PROCESOS ADMINISTRATIVOS

Ejecutar las tareas administrativas como procesos de la aplicación.
Es común tener que realizar tareas administrativas o de mantenimiento en una aplicación, tales como ejecutar scripts para limpiar datos, exportar datos, o disparar una consola para ejecutar algún código arbitrario.

Este tipo de procesos administrativos se deberían manejar en un ambiente idéntico al de los procesos normales de la aplicación. Es decir, son parte del código base de la aplicación y se ejecutan en el contexto de una aplicación liberada, con la misma configuración. También se debe utilizar la misma técnica de aislamiento de dependencias (ej. bundle exec o Virtualenv) que se haya usado con los procesos normales de la aplicación.

CONCLUSIÓN

Como arquitectos y desarrolladores debemos entender la diferencia entre una aplicación que puede funcionar en la nube y otra que está diseñada para aprovechar la nube. Los lineamientos descritos aquí brindan una guía de cómo lograr lo último.

Si quieres conocer más al respecto, te invito a revisar el manifiesto completo [1].

Referencias

[1] "The Twelve-Factor App". https://12factor.net

UN VISTAZO A LA ARQUITECTURA SERVERLESS

—

Por Pedro Galván

Uno de los temas que hemos explorado durante los últimos meses en los distintos canales de Software Guru, es el de arquitectura serverless (sin servidor). Ya en el primer episodio del vlog Devotion lo comentamos de forma casual, y luego en el congreso SG Next tanto Verónica López como Obie Fernández nos platicaron al respecto con mayor detalle. Así que ahora es mi turno de compartir mi perspectiva sobre este modelo arquitectónico.

Lo primero que hay que aclarar es que, a pesar de lo que diga su nombre, en las arquitecturas serverless sí hay servidores. Y de hecho no es uno, sino muchos. La diferencia respecto a las aplicaciones tradicionales es que no hay un servidor maestro que controle el flujo de la aplicación, sino que los servidores simplemente hospedan servicios que atienden peticiones de forma atómica y no están conscientes del flujo de la aplicación, ya que éste se controla del lado del cliente. Dichos servicios pueden haber sido implementados por nosotros (en el caso de comportamiento específico a nuestra aplicación) o también podemos usar servicios proporcionados por terceros para resolver aspectos comunes tales como, autenticación de usuarios, almacenamiento, mensajería. Por último, los desarrolladores no tienen ninguna visibilidad a los servidores que hospedan los servicios, ya que estos son administrados por terceros. Así que los desarrolladores se pueden dedicar exclusivamente a implementar la funcionalidad de la aplicación sin tener que preocuparse por administrar servidores. La figura 1 brinda un panorama de este concepto.

Las aplicaciones serverless típicamente hacen uso extensivo de servicios terceros para cumplir tareas que no son centrales o específicas a la aplicación. Una aplicación puede utilizar una familia de servicios terceros de un mismo proveedor (ej. Firebase) o usar servicios de proveedor distintos, por ejemplo una aplicación podría usar Auth0 para la autenticación de usuarios, Azure Media Services para entregar video por streaming, y OneSignal para push notifications.

Una de las principales responsabilidades de una aplicación tradicional centrada en el servidor es controlar el ciclo de petición-respuesta. Los controladores en el servidor procesan las entradas, invocan el comportamiento correspondiente y construyen una respuesta dinámica usando un motor de plantillas (*template engine*). En contraste, en una aplicación serverless, donde el comportamiento se ensambla a partir de servicios, el control del flujo se hace del lado del cliente y el contenido se genera de forma dinámica en el cliente. Por ejemplo, las aplicaciones móviles, IoT e incluso las de smart TV utilizan frameworks de interfaz de usuario para generar contenido dinámico a partir de información obtenida por medio de llamadas a APIs remotas.

En las aplicaciones centradas en el servidor, la aplicación está

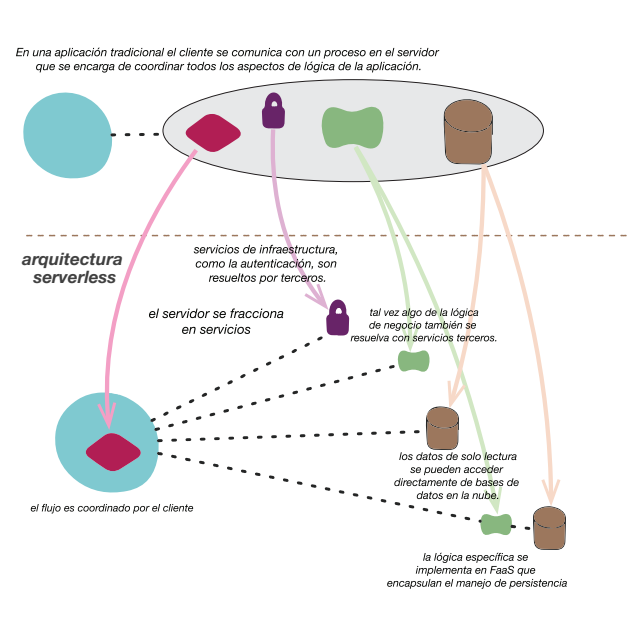


Figura 1. Arquitectura serverless. Fuente: http://martinfowler.com [1]

compuesta por procesos que están continuamente activos y atendiendo peticiones. En contraste, en las aplicaciones serverless el ciclo de vida de los programas es mucho más corto, tendiendo a ser un solo ciclo petición/respuesta en HTTP. El programa se activa cuando recibe una petición, la procesa y en cuanto termina se apaga. Este tipo de programas típicamente están hospedados en un entorno como Amazon Lambda, Azure Function o Google Cloud Functions, que se encarga de gestionar el ciclo de vida y escalabilidad del programa. Este tipo de estructura se conoce como "Funciones como Servicio" (FaaS).

VENTAJAS Y DESVENTAJAS

Cualquier decisión de diseño arquitectónico involucra ventajas y desventajas. Uno de los principales beneficios de la estrategia serverless es el ahorro en costo. En sistemas que tienen patrones de tráfico con picos repentinos, tener un servidor con grandes recursos continuamente disponible es costoso; en cambio, en el modelo FaaS se cobra por petición atendida. Otro componente de costo que te ahorras con serverless es el del personal y del esfuerzo requerido para administrar el servidor.

En cuanto a las desventajas, la primera sería la complejidad que se agrega al estructurar las aplicaciones de esta forma. También puede haber un impacto en el desempeño de la aplicación debido a la carga y latencia introducida al estar continuamente invocando servicios que no están inmediatamente disponibles.

Este artículo apenas ha sido un panorama general sobre serverless. Si te interesa conocer más al respecto, te recomiendo leer el artículo "Serverless Architectures" [2] de Mike Roberts.

Referencias

[1] B. Janakiranam. "Serverless". http://swgu.ru/rs
[2] M. Roberts. "Serverless Architectures". http://swgu.ru/rr

ARQUITECTURA LAMBDA

COMBINANDO LO MEJOR DE DOS MUNDOS

—

Por Benjamín Vera-Tudela

A alto nivel hay dos tipos de procesamiento de datos: el primero es el procesamiento de datos en modo batch, el segundo es en modo stream o tiempo (semi)-real. Elaborando un poco más sobre esta diferencia, el procesamiento batch es aquel que nos permite procesar volúmenes de datos en tiempos espaciados (ej. cada 15 minutos, cada 3 horas, o diario). Mientras que el modo stream es aquel que nos permite procesar datos casi al instante en que estos son producidos (ej. cada 100 milisegundos o cada segundo).

Para entender por qué esta diferencia es importante, podemos utilizar ejemplos de la vida real que la hacen más evidente. Por ejemplo, si una empresa quiere entender al final de cada trimestre cuál fue la tendencia de venta de un producto, para tomar decisiones, es necesario almacenar datos sobre las transacciones de venta del producto a lo largo del mismo periodo para luego realizar el procesamiento. Si el volumen de datos es alto, este procesamiento podría tomar varios minutos o incluso horas, y en este caso en particular es probable que quienes tomen decisiones estén dispuestos a esperar ese tiempo para poder hacerlo.

Sin embargo en otros casos, la espera para tomar decisiones puede ser costosa, y por lo tanto se necesita hacerlo de manera casi inmediata. Por ejemplo, si se desea bloquear el uso de una tarjeta de crédito que aparenta ser fraudulenta, no podemos esperar varios minutos ya que el daño realizado por ese tipo de transacciones puede ser elevado. En estos casos es mejor tomar decisiones casi al instante en que la información de una posible transacción fraudulenta esté disponible.

Cuando los volúmenes de datos en cualquiera de estos casos son muy altos —es decir, cuando afrontamos temas de big data como es el caso hoy para muchas empresas operando en la era digital, y como lo será en los próximos años con el internet de las cosas— la complejidad y el costo de implementar sistemas de información que nos permitan manejar ambos tipos de procesamiento también incrementa. Es ahí en donde nace el concepto de la Arquitectura Lambda.

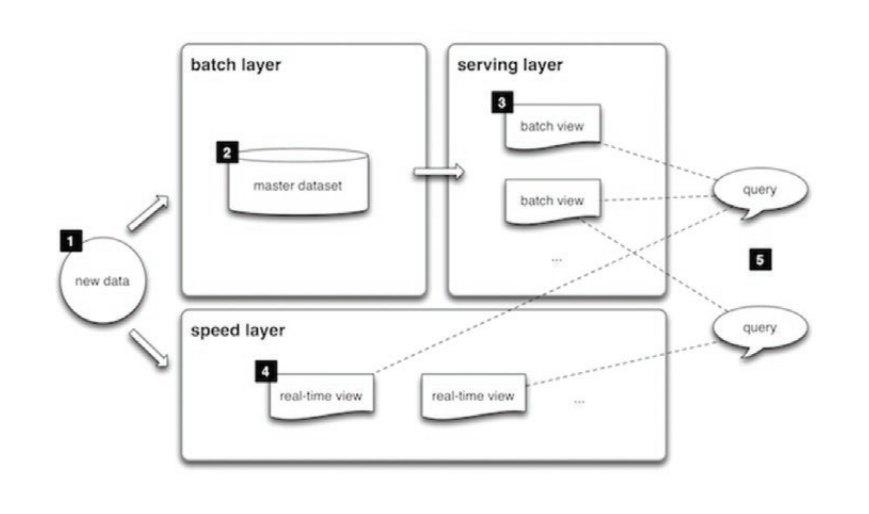


Figura 1. Arquitectura Lambda.

En una arquitectura lambda la idea es implementar sistemas de información que combinan ambas modalidades de procesamiento de datos: batch y stream. Esto nos da lo mejor de dos mundos, ya que el modo batch nos brinda un alcance completo y confiable mientras que el modo stream nos da los datos en línea para decisiones instantáneas.

La figura 1 muestra un modelo de cómo funciona la arquitectura lambda. Los datos que entran al sistema se despachan tanto a la capa batch como a la capa de velocidad (speed). La capa batch escribe los datos al master data set y prepara las vistas batch, pasándolas a la capa de servidor. Esta última se encarga de indexar las vistas batch de manera que pueda responder a búsquedas con muy baja latencia. El problema es que el proceso de escribir datos y luego indexarlos es lento, por lo que éstos no están disponibles de forma instantánea; es aquí donde entra en acción el rol de la capa de velocidad; que se dedica a exponer solamente los datos más recientes, sin preocuparse por escribirlos a un registro permanente. El resultado de cualquier búsqueda puede conjuntar datos provenientes tanto de vistas de la capa batch como de la capa de velocidad.

TECNOLOGÍAS INVOLUCRADAS

En cada una de las capas de esta arquitectura se utilizan tecnologías especializadas para cada propósito. Aunque es posible utilizar distintas opciones, una opción popular es utilizar Apache Sqoop + HDFS + Hive para capturar, almacenar y procesar datos en forma batch, y Apache Kafka + HBase + Spark para capturar, almacenar y procesar datos en forma stream. Existen proveedores —como Cloudera, donde yo colaboro— que ofrecen productos que integran estas herramientas y permiten fácilmente incorporar arquitecturas lambda.

Referencias

[1] http://lambda-architecture.net

BIO. **Benjamin Vera-Tudela** es Senior Big Data Architect en Cloudera, compañía líder en el desarrollo y comercialización de tecnologías abiertas para big data.

PRERREQUISITOS PARA UNA IMPLEMENTACIÓN EXITOSA DE LA ENTREGA CONTINUA

Por Cyrille Le Clerc

La entrega continua está ganando rápidamente la atención de las organizaciones, ya que ayuda a satisfacer la demanda creciente para entregar software de manera ágil. Con su énfasis en mantener el software en un estado "listo para despliegue" en todo momento, es una evolución natural de las prácticas de integración continua y desarrollo ágil de software. Sin embargo, los retos culturales y operacionales para lograrlo son mucho más grandes que dichas prácticas. Para la mayoría de las organizaciones, la entrega continua requiere la adaptación y extensión de los procesos existentes de liberación de software. Los roles, relaciones y responsabilidades de la gente en toda la organización también pueden ser impactados; por otra parte, las herramientas utilizadas para entregar, actualizar y mantener el software deben apoyar la automatización y colaboración en forma apropiada, con el propósito de minimizar demoras en el despliegue de nuevos servicios y proporcionar ciclos de retroalimentación rápida en toda la empresa.

Las organizaciones que están buscando hacer la transición a la entrega continua deben considerar los siguientes siete prerrequisitos. Son pasos prácticos que les permitirán ejecutar en forma exitosa los cambios culturales y operacionales dentro del marco legal y funcional en el cual operan.

LOS EQUIPOS DE DESARROLLO, QA Y OPERACIONES DEBEN TENER METAS COMPARTIDAS Y COMUNICARSE

Mientras que la integración continua limita el alcance del equipo de desarrollo, la entrega continua incluye las fases de prueba del equipo de aseguramiento de calidad (QA) y los despliegues a producción manejados por el equipo de operaciones. Esto es un cambio de paradigma importante, y para tener éxito evolucionando una práctica de integración continua a entrega continua, es crítico que los equipos de desarrollo, QA y operaciones tengan un modelo de gobernanza (*governance*) compartido. La colaboración y la comunicación son componentes vitales del desarrollo exitoso de software hoy en día, y en un ambiente de entrega continua tienen que tomar el centro del escenario.

LA INTEGRACIÓN CONTINUA DEBE FUNCIONAR ANTES DE MOVERSE A LA ENTREGA CONTINUA

BIO. **Cyrille Le Clerc** es Director of Product Management en la empresa CloudBees. Previamente Cyrille fue CTO de Xebia. Fue uno de los primeros proponentes del modelo ágil de desarrollo "You Build It, You Run It" , que ha puesto en marcha con una gran variedad de clientes.

La entrega continua es una extensión de la integración continua, su prerrequisito es tener la integración continua implementada y funcionando, incluyendo control de versiones, y construcción automatizada (*automated builds*) con pruebas unitarias y pruebas automatizadas.

AUTOMATIZA Y HAZ VERSIONES DE TODO

La entrega continua involucra la repetición continua de muchas tareas tales como hacer builds, desplegar aplicaciones y configuraciones, restaurar ambientes y bases de datos. Todas estas tareas se deben automatizar con herramientas y scripts, y todo se debe mantener bajo control de versiones para poder auditarlo y reproducirlo.

COMPARTIR LAS HERRAMIENTAS Y PROCEDIMIENTOS ENTRE EQUIPOS ES CRÍTICO

La entrega continua apunta a validar los procedimientos de implementación y automatización utilizados en el ambiente de producción; para hacerlo exitosamente, estos procedimientos y automatizaciones deben ser utilizados tan temprano como sea posible de forma tal que, sean ampliamente probados cuando se usen para implementar software a el ambiente de producción. En la mayoría de los casos, las mismas herramientas pueden utilizarse en todos los ambientes (pruebas, preproducción y producción).

Los scripts de automatización deben manejarse en repositorios compartidos de código fuente de tal forma que cada equipo —desarrollo, QA y operaciones— pueda mejorar las herramientas y los procedimientos. Mecanismos como *pull-requests* pueden ayudar al gobierno de estos scripts y herramientas.

LA APLICACIÓN TIENE QUE SER AMIGABLE EN SU DESPLIEGUE PARA HACER QUE LAS IMPLEMENTACIONES NO GENEREN EVENTOS DISRUPTIVOS

Las aplicaciones deben simplificar sus procedimientos de instalación y marcha atrás (*rollback*) para minimizar conflictos.

Un paso importante para lograrlo es reducir el número de componentes y de parámetros de configuración. La facilidad de dar marcha atrás, brinda la posibilidad de que en caso de que haya algún problema con una instalación se pueda rápidamente deshacer y regresar el sistema a su estado anterior. La capacidad de activar capacidades de forma independiente (*feature toggle*) permite desacoplar la instalación de binarios de la activación de capacidades, de esta manera si hay un problema al activar una capacidad, simplemente se desactiva en lugar de tener que reinstalar los binarios anteriores. Una marcha atrás entonces puede ser simplemente la desactivación de la misma , gracias a una tecla de conmutación.

Se debe poner especial atención a cualquier cambio en el esquema de las bases de datos, ya que se pueden volver mucho más complejas las implementaciones y los retrocesos. El patrón de diseño sin esquema de las bases de datos NoSQL trae mucha flexibilidad, pasando la responsabilidad del esquema desde la base de datos, al código. Este concepto también puede ser aplicado a bases de datos relacionales.

LA INFRAESTRUCTURA DEBE APUNTALAR EL PROYECTO PARA POTENCIAR A LA GENTE Y EQUIPOS

Las infraestructuras deben proporcionar todo el equipamiento (GUIs, APIs y SDKs) y documentación requeridos para empoderar al equipo de desarrollo y QA, y hacerlos autónomos en su trabajo. Estas tareas incluyen:

- Poder implementar la versión de su elección de la aplicación.
- Gestionar los parámetros de configuración (ver, modificar, exportar, importar).
- Gestionar bases de datos (crear snapshots y restaurar a partir de estos).
- Dar acceso a las bitácoras (logs) de la aplicación.

Las plataformas de nube pública, principalmente en modelos de Plataforma como Servicio (PaaS), son ejemplos de plataformas amigables con los proyectos.

LAS VERSIONES DE APLICACIONES DEBEN ESTAR LISTAS PARA SER ENVIADAS A PRODUCCIÓN

Una de las metas más importantes de la entrega continua es permitir que el dueño de producto pueda desplegar en producción cualquier versión de la aplicación que pase exitosamente el conducto (*pipeline*) de entrega continua, en lugar de solo poder liberar las versiones mayores que resultan al final de una iteración.

Alcanzar este objetivo requiere cambios importantes en la forma en que son diseñadas las aplicaciones, por ejemplo: las características que aún no han sido validadas por el equipo de QA se deben poder ocultar a los usuarios finales sin que la base de código se modifique. Dichas características deben estar en ramas (*branches*) de código distintas a la rama maestra, e incorporarse a la rama maestra hasta que hayan sido validadas por QA y aceptadas para su uso en producción. Otra posibilidad es usar feature toggling que consiste en poder activar o desactivar una funcionalidad desde alguna consola de administración.

Las herramientas de construcción deben evolucionar del concepto de versiones semánticas (*SemVer*) hacia un flujo continuo de versiones. En el caso de subversion, es sencillo ya que automáticamente provee un número de versión ordenado. En el caso de Git, dado que su árbol de versiones no es lineal, es un poco más complicado y requiere acuerdos en cuanto a cómo se va a organizar el árbol de versiones y la nomenclatura que se le dará.

CONCLUSIÓN

La entrega continua no solo se trata de un conjunto de herramientas. También debe tomar en cuenta a las personas y la cultura organizacional. La tecnología, la gente y el proceso necesitan estar alineados para hacer que la entrega continua sea exitosa; y un enfoque colaborativo es fundamental para su éxito. Implementar estas mejores prácticas puede permitir a las organizaciones cosechar las recompensas de un enfoque automatizado más fluido al desarrollo de software, y uno que también proporciona agilidad de negocios.

ES EL FUTURO

Por Paul Biggar

Este artículo es una versión editada y traducida al español de la nota "It's the future" publicada por Paul Biggar en el blog de la empresa CircleCI. La nota original está disponible en https://circleci.com/blog/its-the-future/

Hola. El gerente me dijo que hablara contigo porque eres experto en aplicaciones web.

Bueno, en realidad ya no hago web, ahora hago sistemas distribuidos. Justo acabo de regresar de ContainerCamp y Gluecon y voy a DockerCon la próxima semana. Estoy muy emocionado por cómo están avanzando las cosas, haciendo todo más sencillo y confiable. ¡Es el futuro!

Ok. Solo quiero hacer una aplicación web sencilla de altas, bajas, cambios. Estaba pensando hacerla en Ruby on Rails y ponerla en Heroku. ¿Está bien?
Oh no. Eso ya es viejo. Heroku está muerto, ya nadie lo usa. Ahora debes usar Docker. Es el futuro.

Ok. ¿Y qué es eso?
Docker es la nueva forma de usar contenedores. Es como LXC pero también es un formato de empacado, plataforma de distribución y herramientas para hacer sistemas distribuidos de forma sencilla.

¿Contenedores? ¿LXE?
LXC, es como chroot pero mejor.

¿cher-oot?
Ok, mira ... Docker, contenedores, es el futuro. Es como virtualización pero más rápida y barata.

Ah, como Vagrant.
No, Vagrant está muerto. Lo que debes usar son contenedores, es el futuro.

Ok, ¿entonces no necesito saber nada sobre virtualización?
Bueno, todavía necesitas virtualización porque los contenedores todavía no resuelven todos los aspectos de seguridad.

Ok, estoy un poco perdido. Vamos con calma. Entonces, los contenedores son parecidos a la virtualización. ¿Puedo usarlos en Heroku?
Bueno, Heroku tiene algo de soporte para Docker, pero ya te dije: Heroku está muerto. Lo que debes hacer es ejecutar tus contenedores sobre CoreOS.

¿Qué es CoreOS?
Es un sistema operativo host (anfitrión) que puedes usar con Docker. De hecho no necesitas Docker, puedes usar rkt.

¿Rocket?
No, "rock-it". Es un formato de contenedores que brinda mejor composición que Docker.

¿Y eso es bueno?
Por supuesto. La composición es el futuro.

¿Y cómo lo usas?
No sé. Casi nadie lo usa todavía.

Hhmm. ¿Decías algo de CoreOS?
Sí. Es un host OS que usas con Docker.

¿Qué es un Host OS?
Lo que ejecuta tus contenedores.

¿Ejecuta mis contenedores?
Sí. Necesitas algo que corra tus contenedores. Por ejemplo, creas una instancia de EC2, le pones CoreOS, corres el demonio de Docker, y entonces le puedes poner imagenes de Docker para que las corra.

¿Y qué parte de eso es el contenedor?
Todo. Mira, tomas tu app, escribes el Dockerfile, generas la imagen localmente y entonces puedes mandar eso a cualquier servidor de Docker.

¿Como Heroku?
Heroku no. Ya te dije que está muerto. Ahora tú operas tu propia nube usando Docker.

¿Qué?
Es fácil. Busca #gifee

¿Gify?
Gifee: Google's infrastructure for everyone else. Básicamente usas tecnologías open source para construir una infraestructura que funcione similar a la de Google.

Para eso, mejor uso el Google Cloud Platform, ¿no?
¿Estás seguro de que siga operando en 6 meses?

Ok, ¿entonces no hay alguien más que opere esto? No quiero tener que operar mi propia infraestructura.
Bueno, Amazon tiene ECS pero debes configurarlo en XML o algo raro.

¿Y OpenStack?
Ew (asco).

En realidad no quiero operar infraestructura.
No te preocupes. Es muy sencillo. Solo creas un cluster de Kubernetes ...

¿Necesito un cluster?
Un cluster Kubernetes. Se encarga de gestionar el despliegue de tus servicios.

¿Servicios? Solo tengo un servicio.
¿De qué estás hablando? Tienes una aplicación, así que deberías tener mínimo unos 10 servicios.
No, solo tengo una app, servicio o lo que sea. Solo uno.
¿No has oído de microservicios? Es el futuro. Tomas tu aplicación monolítica y la divides en muchos servicios, uno por cada tarea que hace tu app.

Eso me parece excesivo.
Es la única forma de asegurar que es confiable. Por ejemplo, si tu servicio de autenticación cae.

¿Servicio de autenticación? Solo planeaba usar una gema de Ruby para user/password que ya he usado en otras apps.
Ok, usa la gema. Pero ponla en su propio contenedor y exponla por medio de un API REST. Así otros servicios pueden usarla y si hay alguna falla, pueden recuperarse de ésta.

Ok, ¿y una vez que tenga decenas de servicios qué sigue?
Ah, sí. Te mencioné Kubernetes, te permite orquestar todos tus servicios.

¿Orquestarlos?
Sí. Tienes tus servicios y necesitas que toleren fallas, así que requieres tener varias copias de ellos. Kubernetes se asegura de que tengas suficientes y que están distribuidos en distintos nodos en tu flotilla, para que esté siempre disponible.

¿Necesito una flotilla de servidores?
Sí, para confiabilidad. Pero Kubernetes la maneja. Y Kubernetes funciona, porque Google lo hizo y corre sobre etcd.

¿Qué es etcd?
Es una implementación de RAFT.

OK, ¿y qué es Raft?
Es como Paxos.

Dios mío, ¿qué tan profundo llega el agujero? Lo único que quiero es lanzar una app *respira profundo*. OK, ¿qué es Paxos?
Es un protocolo de cómputo distribuido de los 70 que nadie entiende ni usa.

Ok, gracias por nada. ¿Y entonces qué con Raft?
Bueno, como nadie entiende Paxos, un chavo que se llama Diego ...

Ah, ¿lo conoces?
No, trabaja en CoreOS. En fin, Diego creó el protocolo Raft para su tesis doctoral porque Paxos es muy complicado. Y entonces escribió etcd como una implementación de Raft. etcd es un almacén distribuido de llave-valor (key-value).

Ah, como Redis.
No. Redis pierde la mitad de sus escrituras si la red se particiona. Por eso etcd es distribuido.

OK, ¿y de qué me sirve?
Kubernetes crea por default un cluster de cinco nodos usando etcd como bus de mensajes. Esto se combina con los propios servicios de Kubernetes para brindar un sistema de orquestación tolerante a fallas.

¿Cinco nodos? Tengo una app. ¿Cuántas computadoras necesito para todo esto?
Bueno, vas a tener como unos 12 servicios, y necesitas copias redundantes de cada uno, balanceadores de *carga, el cluster de etcd, tu base de datos, el cluster de Kubernetes, así que digamos que unos 50 contenedores.*

Eso es una locura.
No te preocupes. Los contenedores son muy eficientes, así que deberías poder distribuir esos 50 contenedores en unas ocho computadoras. ¿No es maravilloso?

Sí, claro. ¿Y con todo esto, ya podrá desplegar mi app?
Claro. Bueno, el almacenamiento todavía no está del todo resuelto con Docker y Kubernetes, y la configuración de la red es un poco latosa, pero se resuelve.

Ok. Déjame repetir para ver si entendí: entonces, necesito dividir mi simple aplicación en unos 12 microservicios, cada uno con su propia API, y cada uno en su contenedor de Docker, levantar una flotilla de 8 máquinas que son anfitriones de Docker corriendo CoreOS, orquestarlos usando un cluster de Kubernetes corriendo etcd, resolver los detalles de almacenamiento y red, y luego conforme creo nuevas versiones de cada servicio debo desplegarlos de forma continua en los nodos de mi flotilla. ¿Eso es todo?
Sí, ¿no es fabuloso?

Me regreso a Heroku.

El Desarrollo de Baja Codificación ya no es Opcional

Por Aad van Schetsen

El desarrollo de aplicaciones con baja codificación (*low-code development*) no es nada nuevo. En muchos sentidos, es simplemente una rama de la tendencia 4GL de las décadas los 80 y 90. Pero el mundo de la tecnología ha cambiado y hoy en día, la baja codificación se ha vuelto más relevante que nunca.

De acuerdo con Forrester, el mercado mundial para el desarrollo con baja codificación eclipsará los 10 mil millones de dólares en 2019. Eso tiene sentido, ya que las fortalezas de dicha estrategia —iteraciones rápidas, colaboración simple, y fácil mantenimiento a largo plazo— se complementan perfectamente con las necesidades de las empresas que requieren llevar de manera eficiente aplicaciones móviles comerciales, de escritorio, de nube y arquitecturas cliente-servidor.

En pocas palabras, si tienes que competir en la creciente industria del software, es necesario el desarrollo con baja codificación. A continuación comparto cinco razones de por qué hacerlo:

COLABORACIÓN CASI EN TIEMPO REAL ENTRE LAS ÁREAS DE NEGOCIO Y TI

Las aplicaciones son hechas por desarrolladores, no necesariamente para desarrolladores. No importa qué tan elegante sea la interfaz de usuario, o qué tan intuitiva sea la experiencia de usuario, una aplicación que no concuerda con los objetivos de negocio es, en cierto nivel, un fracaso.

Afortunadamente, aquí es donde el desarrollo con baja codificación brilla en particular. Debido a que la cantidad de codificación manual de bajo nivel se reduce drásticamente, el desarrollo avanza muy rápido. Lo que le lleva a un desarrollador días o semanas programar manualmente, puede en muchos casos realizarse en horas con herramientas de baja codificación. Como resultado, los stakeholders pueden ver su visión tomar forma rápidamente, dejando tiempo para ajustes rápidos. Los desarrolladores y los usuarios de negocios disfrutan de una colaboración en tiempo casi real y del intercambio de ideas, en lugar de la metodología tradicional iterativa de "prueba y error".

Adicionalmente, en el desarrollo de baja codificación basada en modelos, la aplicación es derivada directamente de las reglas del negocio. El comportamiento de la aplicación no está impulsada solo por el código sino por la misma lógica del negocio. En tal escenario, los requerimientos del negocio y la aplicación están íntimamente ligados; la aplicación debe ajustarse a igualar las reglas de negocio más exigentes, ya que están inextricablemente "construidas al mismo tiempo".

AUMENTO DEL ENFOQUE DEL DESARROLLADOR HACIA LA FUNCIONALIDAD

Las empresas contratan trabajadores expertos con altos conocimientos esperando que sean eso —expertos en su área. Cuando se trabaja en una plataforma de desarrollo con baja codificación, los programadores pasan menos tiempo "escribiendo" genérico y más tiempo creando objetos que representan la funcionalidad de muchas líneas de código. Esto aumenta considerablemente la productividad potencial de cada desarrollador, se consiguen resultados más rápido y se mejora la utilización de sus habilidades únicas.

DESPLIEGUE TRANSPARENTE, INDEPENDIENTE DE PLATAFORMA

En un proyecto de baja codificación, los desarrolladores trabajan en la definición de modelos y se centran en la interfaz y experiencia de usuario, en lugar de preocuparse acerca de cómo se implementará la aplicación a través de diferentes arquitecturas. El despliegue de una aplicación con baja codificación es completamente transparente; no importa el paradigma o plataforma que se utilice —nube, servidor, web, móvil, cualquiera que sea— todos los datos se extraen de localidades específicas en el modelo, con la configuración necesaria y con muy poca codificación adicional.

Los desarrolladores pueden concentrarse en hacer una genial aplicación dentro del contexto de los requisitos de negocio y no tener que preocuparse por el entorno de implementación, ya que muchas de las tareas de bajo nivel que participan en el despliegue de una aplicación orientada a objetos son eliminadas.

ADECUADA PARA EL USO A LARGO PLAZO

La velocidad y facilidad con la que una aplicación de baja codificación puede ser modificada no termina en el despliegue. Puesto que tales aplicaciones siguen siendo muy fáciles de mantener —ya que los ajustes del modelo afectan a muchos elementos individuales— es especialmente adecuado para el uso a largo plazo. Añadir soporte para una nueva tecnología o plataforma es simple, y los requisitos de negocio cambiantes se pueden resolver eficientemente de manera extraordinaria.

UNA SOLA BASE DE CÓDIGO CON MENOR MARGEN DE ERROR

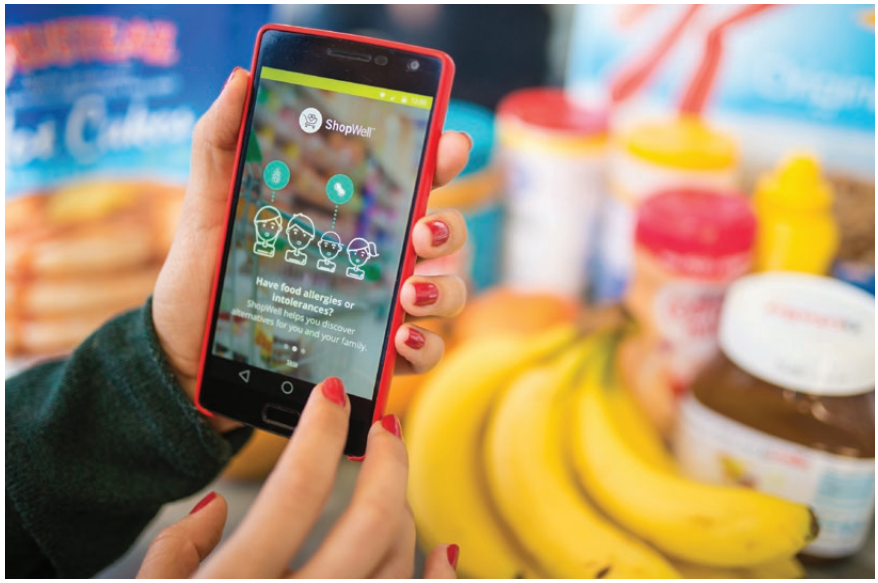
En el desarrollo tradicional de aplicaciones, es común ver una disminución gradual en la calidad y facilidad de uso. La primera iteración se ve muy bien, pero luego hay ajustes y cambios en las necesidades de los usuarios. Pasa otra iteración y sucede lo mismo. Conforme se acerca la fecha de entrega, la aplicación comienza a parecer un muégano, adiciones tardías que se aplican a las prisas, y con distintos criterios de calidad. Esto se ve agravado por el hecho de que muchos cambios de última hora afectan componentes centrales de la aplicación, creando una situación donde las funciones más importantes tienen la menor calidad.

En contraste, en el desarrollo con baja codificación, los desarrolladores trabajan en una simple línea base de código limpio. Las modificaciones se realizan una vez, y se extienden a lo largo del desarrollo de la aplicación, debido a que el proceso general es más sencillo. Los usuarios de negocio pueden proporcionar retroalimentación temprana durante el proceso, multiplicando el beneficio. La modificación de los requisitos y especificaciones se pueden abordar de forma cuidadosa y metódica, en lugar de manera previa o posterior al lanzamiento.

CONCLUSIÓN

Mediante la eliminación de código de bajo nivel que se programa de manera manual, se aumenta el tiempo que los desarrolladores dedican a tareas de alto valor. El desarrollo con baja codificación permite a los equipos producir rápidamente aplicaciones que se alinean estrechamente con los objetivos de negocio. El despliegue —sin importar la plataforma— es simple y consistente, y las aplicaciones conservan un alto nivel de calidad pensando a futuro. No es la respuesta a todos los retos de programación —siempre habrá un lugar para Java, .NET y otros frameworks orientados— pero el desarrollo con baja codificación está encontrando cada vez más su lugar dentro de los equipos de desarrollo modernos.

Aad van Schetsen es CEO de la empresa Uniface. Cuenta con tres décadas de experiencia en la industria, con un liderazgo demostrado y profundo conocimiento en negocios empresariales, administración general, software y TI.



Cómo Aumentar la Lealtad de tus **Usuarios en Aplicaciones Móviles**

Por Misael León

Hablemos de cuando los usuarios abandonan nuestra aplicación y no vuelven más. Ese duro momento que nos duele tanto a los equipos de productos. Si está sucediendo con tu app no te sientas solo; es de hecho un suceso promedio en la industria de software y es tan alto que es impresionante.

Los datos duros dicen: la media de las aplicaciones en el mercado pierde 77% de sus usuarios después de 3 días de haberla instalado. Dentro de 90 días sube a 95% [1]. El terror encarnado. Pero, ¿qué hacer al respecto? A continuación veamos las causas más comunes y algunas soluciones para ponerle fin a la pesadilla.

CAUSAS COMUNES DE ABANDONO

Hay distintas razones para que un usuario abandone una aplicación. Exploremos una de las más recurrentes, cuando creamos una barrera inicial justo después de haberla instalado.

1. UN PROCESO DE ONBOARDING CONFUSO

Onboarding es la actividad de educar a los usuarios respecto a cómo funciona nuestra aplicación o tal vez sobre los beneficios de utilizarla. Son una serie de pantallas informativas colocadas al inicio de la interacción.

Usualmente nuestra intención de educar es genuina. Sin embargo, es común abusar de este recurso y presentar mucha información en una sola pantalla, o ceder a la tentación de hablar de todas las funcionalidades que nuestro producto es capaz de hacer. Esto ocasiona que nuestros usuarios terminen sintiéndose abrumados y confundidos. Recuerda que lo que ellos quieren es experimentar tu aplicación de inmediato, usar el juguete nuevo. Déjalos descubrirla por sí mismos.

Solución. Realiza entrevistas con algunos de tus usuarios (entre 8 y 10 será suficiente) para entender cómo tu producto encaja en su estilo de vida. De esto aprenderás a identificar qué los motiva y qué los frustra. También podrás conocer sus hábitos personales y de consumo. De esta manera podrías seleccionar el tipo de onboarding process adecuado. A continuación describo los tipos de onboarding más comunes:

- Orientado a beneficios: es cuando describimos en esas pantallas iniciales la transformación que el usuario experimentará si utiliza nuestra aplicación. Tal vez será más feliz, más saludable, más conectado con sus amigos, más organizado. Quizás ganará acceso a millones de canciones o recomendaciones de recetas. Estos son beneficios subjetivos. Recomiendo no ir más allá de 4 o 5 pantallas.

- Orientado a funciones: aquí es cuando describimos lo que la aplicación hace propiamente. Tal vez permitirá crear listas de reproducción inteligentes, compartir información en redes sociales u obtener recomendaciones de expertos. Estos son beneficios objetivos que tu producto brindará. Recuerda que el usuario ya está convencido de utilizarla, dado que ya la ha instalado. Así que ten máximo 3 o 4 pantallas y úsalas para sorprenderlos con algo inesperado.

Misael León (@misaello) es un UX Designer que trabaja en Nearsoft investigando usuarios, desarrollando ideas de productos y diseñando prototipos. Su misión es la de crear herramientas intuitivas para que otros puedan realizar su trabajo. Le apasiona difundir las mejores prácticas de UX en comunidades de diseñadores y desarrolladores.

• Orientado a funcionalidades específicas: cuando tu producto funciona de manera diferente al estándar del mercado o cuando es algo radicalmente nuevo, lo ideal es indicar dónde están ciertas funcionalidades y explicar cómo usarlas. Es decir, en dónde los usuarios podrán crear listas de reproducción, cómo podrán exportar información o cómo sacar mayor provecho de alguna búsqueda tal vez. Es ideal utilizar entre 3 a 4 pantallas. Dejemos que las personas gocen de su deseo innato de descubrir lo nuevo por sí mismas.

No existe una solución única que sea adapte para todas las aplicaciones. Puedes considerar incluso no utilizar ningún tipo de onboarding en absoluto. Una interfaz intuitiva debería ser capaz de explicarse a sí misma.

2. LLENAR UN PERFIL MUY LARGO

¡Hurra! Tenemos un usuario nuevo, pidámosle todos los datos posibles para conocerlo mejor. Errrrr. Esta es una práctica nociva. Pedirle mucha información a un usuario cuando recién llega es como si entraras a una tienda y te pidieran llenar una encuesta para identificarte. Piensa, ¿realmente necesitas el número teléfonico de inmediato?

Pedir a los usuarios que llenen un perfil muy largo antes que experimenten tu aplicación los hará sentirse atrapados. De nuevo, los ponemos muy lejos del juguete nuevo.

Solución. Lo más sencillo es pedirles hacer login mediante servicios conocidos como Google o Facebook. De no ser posible puedes definir un perfil modelo de tus usuarios. En UX le llamamos Persona [2]. La información base la puedes obtener a través de entrevistas, analizando tus analíticos o realizando encuestas. De esta manera podrás tener una visión aproximada de quiénes son ellos exactamente. Entendiendo sus hábitos y estilo de vida, puedes determinar qué información te es relevante y cuáles datos están dispuestos a proporcionar. La diversidad moderna hace del uso de perfiles una necesidad. Por ejemplo, no podemos tratar de igual manera a una mamá que a un millennial, ambos tienen una visión del mundo muy diferente.

3. LIMITACIONES EN LAS VERSIONES

Imagina que vas a comprar una carro nuevo, de inmediato sientes la adrenalina que experimentarás al manejarlo y mueres por hacer una prueba de manejo. Pero por alguna razón desconocida la agencia solamente te deja encender el carro, pero no manejarlo. ¡Imposible! Esto nunca sucede. Y sin embargo constantemente vemos esta situación en las aplicaciones móviles.

Cuando bloqueamos funcionalidades de nuestro producto, no estamos dejando a los usuarios "probar" la mercancía antes de comprarla. Solamente causamos frustración cuando ponemos a los usuarios en la situación de comprar una versión completa sin antes dejarlos evaluarla.

Soy creyente fiel de los trial versions, por tanto dejar a las personas utilizar una versión completa para sentirse seducidos por las bondades de tu aplicación, hará que suban tus cifras de conversión. Después de todo, nos medimos la ropa antes de comprarla, probamos una muestra de galletas antes de decidirnos por una caja. Nuestro comportamiento en el mundo real no es tan diferente al mundo digital.

Solución. Realiza *A/B testing* [3] para determinar qué incluir en tu trial. Por ejemplo, puedes liberar una versión lite con cierto número de funcionalidades, digamos 4 de 10, y mides el nivel de conversión de compra de la versión completa. Posteriormente, liberas otra versión con 7 de 10 funcionalidades y mides si la conversión mejoró.

4. CONFIGURACIONES INICIALES COMPLICADAS

Creemos saberlo todo. Es por eso que a menudo no leemos los manuales de lo que compramos. Esto es excepcionalmente cierto para los productos digitales. Poner a los usuarios a llenar configuraciones complicadas y robustas antes de dejarlos experimentar tu aplicación es como ponerlos a leer un manual. Lo que termina ahuyentando a cualquiera.

Si tu producto realmente necesita cierto tipo de configuraciones iniciales por parte de tus usuarios, piensa primero cuáles de esas pueden ser predefinidas o cuáles pueden ser completadas en una ocasión posterior.

Solución. Realiza pruebas de usabilidad para determinar en qué parte del proceso de configuración tus usuarios se confunden y abruman. Si pruebas la configuración inicial con 5 personas [4] podrás descubrir los problemas más críticos de usabilidad de tu aplicación. Los datos que obtendrás serán útiles para determinar esos momentos bajos que tu usuario experimenta al ir llegando a tu aplicación. De esta manera podrás determinar más fácilmente qué configuraciones predeterminar, cuáles posponer y cuáles otras deben ser redefinidas para hacerlas más sencillas.

EL LARGO CAMINO DEL ABANDONO A LA LEALTAD

Sin duda, el éxito del producto con los usuarios está determinado desde el día uno de uso. Esto no es tan fácil como parece, requiere de una medición esmerada sobre lo que funciona y lo que no funciona. Es por eso que realizar investigación de usuarios es necesario para entender por qué la gente termina abandonando nuestra aplicación.

Las aplicaciones que hoy en día son exitosas, lo son porque han logrado convertirse en parte indispensable de la vida de las personas; piensa en Facebook o Snapchat.

Con tantas opciones, la competencia entre apps es intensa. Liberar tu aplicación al mercado no es garantía de éxito, ya que es una entre miles. El aprendizaje continuo es lo que hará fuerte a tu equipo de producto. ¿Ya estás listo para salir a investigar?

Referencias

[1] A. Chen. "New data shows losing 80% of mobile users is normal, and why the best apps do better." http://swgu.ru/rj

[2] S. Carter and J. Christ. "Avoiding Bullshit Personas". Presentación: http://swgu.ru/rk Audio: http://swgu.ru/rl

[3] J. Cardello. "Define Stronger A/B Test Variations Through UX Research." Nielsen Norman Group. http://swgu.ru/rm

[4] J. Nielsen. "Why You Only Need to Test with 5 Users." Norman Nielsen. http://swgu.ru/rn

Special Purpose Languages

PARTE 4 LENGUAJES, COMPILADORES, ARQUITECTURA

—

Por Luis Vinicio León Carrillo

Luis Vinicio León Carrillo es Director General y co-fundador de e-Quallity. Fue profesor-investigador en la universidad ITESO. Realizó estudios de posgrado en Alemania, durante los cuales abordó temas relacionados con la prueba de software y los métodos y lenguajes formales.

En el número anterior mostramos la Jerarquía de lenguajes de Chomsky, que se construye considerando de la estructura de las reglas del conjunto R que define cada tipo de lenguaje. Partamos ahora no de las gramáticas, sino de los lenguajes, y apliquemos unas extensiones: llamamos...

• Lenguajes Regulares (o de Tipo 3, o de manera corta Lgs_3) a aquellos que pueden definirse utilizando Gramáticas Regulares, cuyas reglas son de la forma $A \rightarrow B\ d \mid d$ (o menos compleja). *Nota: las reglas podrían ser de la forma* $A \rightarrow B\ d \mid d$ *, pero no se pueden mezclar ambas formas en una misma gramática.*

• Lenguajes Libres de Contexto (o Tipo 2, o Lgs_2) a aquellos que pueden definirse utilizando Gramáticas Libres de Contexto, cuyas reglas son de la forma $A \rightarrow \alpha$ (o menos compleja). Dentro de estos lenguajes existen varias divisiones. Una de las más relevantes es la que diferencia entre los lenguajes determinísticos y los no-determinísticos; un ejemplo de estos últimos es $a^n\ b^n \cup a^n\ b^{2n}$, que como podrán intuir, para procesarlo es necesario que el compilador asociado realice *backtracking*.

• Lenguajes sensibles al contexto (o tipo 1, o Lgs_1) a aquellos que pueden definirse utilizando Gramáticas Sensibles al Contexto, cuyas reglas o son de una de las siguientes formas:

a) $\beta^1\ A\ \beta^2 \rightarrow \beta^1\ \alpha\ \beta^2$ asdfasd (o menos compleja), donde $\alpha \neq \lambda$

b) $\theta \rightarrow \alpha$ (o menos compleja), donde $|\theta| \leq |\alpha|$

• Lenguajes Estructurados por Frases (o Recursivamente Enumerables, o tipo 0 o Lgs_0) a aquellos que pueden definirse utilizando Gramáticas Estructuradas por Frases, cuyas reglas son de la forma $\theta \rightarrow \alpha$ (o menos compleja), donde $\theta \neq \alpha$. Dentro de estos lenguajes Lgs_0 se encuentra la importantísima clase de los Lenguajes Decidibles (o Recursivos, o Lgs_{dec}), que son aquellos para los que es posible construir programas (por ejemplo, compiladores) que siempre terminan y anuncian si la cadena sí es elemento del lenguaje o si no lo es (aunque para ello consuman cantidades considerables de tiempo y memoria). Las reglas con las que se definen tienen la misma forma que los Lgs0. Podemos ver a los Lgs0 como lenguajes semi-decidibles: los programas que los procesan (por ejemplo, compiladores) pueden determinar cuando las cadenas sí son elementos del lenguaje, pero cuando no es así continúan su procesamiento indefinidamente.

• Lenguajes Generales (o Lgs_{gral}) a aquellos que incluyen a todos los anteriores y a los que no son susceptibles de ser definidos con reglas.

En esta jerarquía ocurre (obviando algunas precisiones) que $Lgs_3 \subset\neq Lgs_2 \subset\neq Lgs_1 \subset\neq Lgs_{dec} \subset\neq Lgs_0 \subset\neq Lgs_{grals}$. Esto enuncia con mayor exactitud lo que mencionamos sobre la cantidad de programas ($|N|$) y de lenguajes ($|2^N|$): de todo el universo de lenguajes Lgs_{grals}, solamente para los Lgs_{dec} podemos escribir procesadores que determinan cuando las cadenas sí forman parte del lenguaje y también cuando no es así.

Ejemplos de especificaciones usando gramáticas

Especifiquemos ahora con gramáticas los lenguajes que definimos "algebraicamente" en el número anterior:

L_1 : $a^m\ b^n\ c^i\ d^k$, para $m, n, i, k \geq 0$ e independientes entre sí:

$E \rightarrow \lambda \mid a\ E \mid b \mid b\ F \mid c \mid c\ G \mid d \mid d\ H$

$F \rightarrow b \mid b\ F \mid c \mid c\ G \mid d \mid d\ H$

$G \rightarrow c \mid c\ G \mid d \mid d\ H$

$H \rightarrow d \mid d\ H$

Como pueden observar, se trata de una Gramática Regular. Escribamos lo que se denomina una derivación para $a^2\ b^3\ c^1\ d^0$:

$E \rightarrow a\ E \rightarrow aa\ E \rightarrow aa\ F \rightarrow aab\ F \rightarrow aabb\ F \rightarrow aabbb\ F \rightarrow aabbb\ G \rightarrow aabbbc\ G \rightarrow aabbbc\ H \rightarrow aabbbc\ \lambda \rightarrow aabbbc$

Este tipo de gramáticas son equivalentes a las conocidas expresiones regulares, y como veremos en seguida, son suficientes para especificar los "scanners" de los compiladores, los cuales se encargan del análisis lexicográfico.

El analizador sintáctico suele ser el componente central del procesamiento de un compilador.

L2: $a^m b^i c^i d^m$, para $m, i \geq 0$ que aparecen por pares y están "anidadas":

$E \rightarrow a\,E\,d \mid F$

$F \rightarrow b\,F\,c \mid \lambda$

Este es un ejemplo "clásico" de una Gramática Libre de Contexto. He aquí una derivación para $a^2 b^3 c^3 d^2$:

$E \rightarrow a\,E\,d \rightarrow aa\,E\,dd \rightarrow aa\,F\,dd \rightarrow aab\,F\,cdd \rightarrow aabb\,F\,ccdd$
$\rightarrow aabbb\,F\,cccdd \rightarrow aabbb\,\lambda\,cccdd \rightarrow aabbbcccdd$

Si hacemos $a = (,\ b = \{,\ c = \},$ y $d =)$ tenemos el caso concreto que mencionamos en la revista anterior.

Este tipo de gramáticas son equivalentes a los conocidos grafos de sintaxis, y son suficientes para especificar los *parsers* de los compiladores, los cuales se encargan del análisis sintáctico.

L3: $a^m b^i c^m d^i$, para $m, i \geq 0$, que aparecen por pares y no están anidadas:

1. $S \rightarrow V \mid \lambda$
2. $V \rightarrow a\,V\,Y \mid a\,c \mid X$
3. $X \rightarrow b\,X\,Z \mid b\,Z$
4. $Z\,Y \rightarrow Z\,T_1$, $Z\,T_1 \rightarrow Y\,T_1$, $Y\,T1 \rightarrow Y\,Z$ /* Reglas que abreviaremos como $Z\,Y \rightarrow Y\,Z$ */
5. $b\,Y \rightarrow b\,c$
6. $c\,Y \rightarrow c\,c$
7. $c\,Z \rightarrow c\,d$
8. $d\,Z \rightarrow d\,d$
9. $b\,Z \rightarrow b\,d$

Excepto por el λ en la primera regla, que es indispensable porque el lenguaje genera λ, y que es aceptable porque solo la genera el Símbolo-inicial, la anterior es una Gramática Sensible al Contexto: en las 3 primeras reglas no hay contexto ($\beta_1 = \lambda$ y $\beta_2 = \lambda$, volviéndolas Libres de Contexto); en las restantes ocurre que $\beta_1 = \lambda$ o que $\beta_2 = \lambda$. He aquí una derivación para $a^3 b^2 c^3 d^2$:

$S \rightarrow V \rightarrow aVY \rightarrow aaVYY \rightarrow aaaVYYY$
$\rightarrow aaaXYYY \rightarrow aaabXZYYY \rightarrow aaabbZZYYY$
$\rightarrow aaabbZYZYY \rightarrow aaabbYZZYY \rightarrow aaabbYZYZY \rightarrow aaabbYYZZY$
$\rightarrow aaabbYYZYZ \rightarrow aaabbYYYZZ \rightarrow aaabbcYYZZ \rightarrow aaabbccYZZ$
$\rightarrow aaabbcccZZ \rightarrow aaabbcccdZ \rightarrow aaabbcccdd$

Con este tipo de gramáticas se podrían especificar (al menos buena parte de) los Sistemas de Tipos de los lenguajes de programación, los cuales definen sus aspectos semánticos (de significado, como los del paso de parámetros que mencionamos, o la equivalencia de tipos). Sin embargo, dado que el procesamiento de estas gramáticas es más complejo y costoso en tiempo que el de las Libres de Contexto, y que la mayoría de los constructos de los lenguajes de computación son de este último tipo, los aspectos semánticos también se suelen abordar apoyándose en gramáticas Libres de Contexto y en una Tabla de Identificadores (en la cual suele almacenarse, entre otras cosas, su nombre, clase (si es variable, procedimiento, función, etc.), tipo (v.gr. integer, string, ⟨char, real⟩ → boolean), y dirección).

Lenguajes y Compiladores

Obviando algunas precisiones, un compilador suele tener una estructura como la que se muestra en la figura 1.

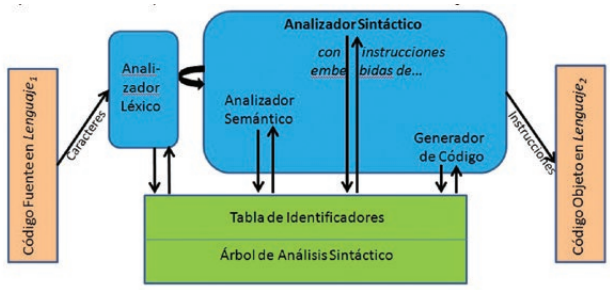


Figura 1. Estructura de un compilador.

Grosso modo, su funcionamiento es como sigue: cada vez que el scanner (analizador léxico) es llamado, reanuda su lectura de caracteres del código fuente escrito en el $Lenguaje_1$; se salta caracteres irrelevantes (como una secuencia de comentarios, blancos, y/o saltos de línea) y luego concatena los que sí son relevantes hasta completar una cadena que tenga significado por sí misma (por ejemplo "{", "main" y "contador" en el lenguaje C); en seguida detecta si la cadena es una palabra reservada (como "main"), un símbolo (como "{",) o un identificador (como "contador"), en cuyo caso lo busca en la Tabla de Identificadores y si no lo encuentra lo da de alta; finalmente, regresa esta información a la subrutina que lo llamó, usualmente el *parser*.

El parser (analizador sintáctico) suele ser el componente central del procesamiento de un compilador. Manda llamar al scanner, el cual le regresa la cadena que acaba de completar y la información que mencionamos arriba. Con esos datos, el parser busca seguir alguna de las reglas gramaticales del lenguaje; en caso de no encontrar alguna envía un mensaje de error. En su procesamiento va utilizando y actualizando la tabla de identificadores (por ejemplo, escribiendo si el Identificador es el nombre de una subrutina o de una variable, así como su tipo), va construyendo el árbol sintáctico del programa en cuestión (ver un ejemplo en seguida), y va ejecutando instrucciones que tiene "embebidas", tanto del analizador semántico que revisa cuestiones relacionadas principalmente con tipos de datos (para lo cual se apoya en la tabla de identificadores), como del generador de código (el cual también se apoya en esa tabla) para ir construyendo el código objeto con instrucciones en el $Lenguaje_2$.

El árbol sintáctico para la cadena $a^2 b^3 c^3 d^2$ del lenguaje L_2, descrito arriba, se muestra en la figura 2. En ella se puede observar que se parte de S0 y se van generando ramificaciones sustituyendo no-Terminales mediante la aplicación de alguna regla de la gramática; las hojas del árbol acaban siendo Terminales o λ.

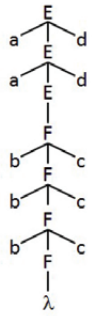


Figura 2. Arbol sintáctico de la cadena $a^2b^3c^3d^2$

Compiladores y Arquitectura

La investigación en arquitectura de software ha estado motivada principalmente porque los sistemas han venido creciendo en tamaño y complejidad, haciendo que el tema de mayor preocupación no sean ya tanto los algoritmos y las estructuras de datos, sino la interacción entre los constituyentes del sistema.

Esta investigación puede verse como una continuación natural del desarrollo de mecanismos de abstracción en los lenguajes de programación, los cuales buscan facilitar el desarrollo de sistemas:

- Los primeros programas de software fueron escritos en lenguaje máquina. La inserción de una nueva instrucción podía requerir tener que revisar manualmente el programa completo, lo cual motivó el desarrollo de ensambladores simbólicos, que hacían actualización automática de referencias y permitían utilizar nemónicos en lugar de códigos de operación de instrucciones. Los posteriores procesadores de macros fueron más allá y permitieron que un solo símbolo fuera sustituido por una secuencia de instrucciones.
- Posteriormente se detectaron patrones de ejecución comunes, como la evaluación de expresiones aritmético-algebraicas, el llamado a subrutinas, así como las instrucciones de alternación (ej. "if") y de repetición (ej. "while").
- Luego se detectaron también patrones en el uso de datos, que condujeron al desarrollo de Sistema de Tipos que incluían tipos básicos y constructores de tipos.
- Después vino la introducción de módulos, que permitió "aislar" subrutinas y tipos de datos para impedir su modificación, y proveer interfaces de los mismos solamente para su uso a (el resto de) los programadores.

Hoy se trabaja en la formalización de patrones arquitectónicos para poder procesarlos con lenguajes formales, facilitando la construcción de la arquitectura los sistemas.

Es común que la arquitectura de un producto de software se defina formalmente como un grafo dirigido definido por: a) un conjunto (finito) de nodos, que son los componentes del producto (subrutinas, datos, etc.); b) un conjunto (finito) de conectores entre los nodos (interacciones como llamadas, accesos a datos, etc.); y (eventualmente) c) un conjunto de restricciones e invariantes sobre los grafos (v.gr. la ausencia de ciclos).

En la literatura especializada se han documentado varios patrones arquitectónicos. Dos de ellos son:

- Pipelines: en ellos los componentes son programas que llamamos filters y que se ejecutan secuencialmente, y los conectores, que llamamos pipes, son los flujos de datos entre ellos. Inicialmente (en los 70's), la arquitectura de un compilador se veía como un pipeline (ver figura 3).
- Repositories: uno de los tipos más importantes de repository son los blackboards, en los cuales hay una(s) estructura(s) de datos central(es) a la(s) que se conectan los componentes, la(s) cual(es) dirige(n) la ejecución de los mismos.

La arquitectura de los compiladores se ha venido sofisticando y hoy en día tiene más bien una estructura de blackboard como la que mostramos al inicio de la sección anterior.

Los compiladores pueden ser un muy buen ejemplo al abordar el tema de los patrones arquitectónicos, ya que facilitan el estudio de la necesidad y aplicabilidad de patrones a la luz de la complejidad y sofisticación el producto en cuestión a lo largo del tiempo. Adicionalmente, cuando se construyen compiladores se aprovechan los avances en el procesamiento de lenguajes formales para generar automáticamente el blackboard y utilizar meta-lenguajes especializados (como como las Expresiones Regulares, la notación BNF, y los Grafos de Sintaxis) con los cuales se especifica la instancia del compilador específico, así como para aplicar el método de análisis sintáctico que resulte más conveniente.

Podremos abundar sobre este tema en números posteriores.

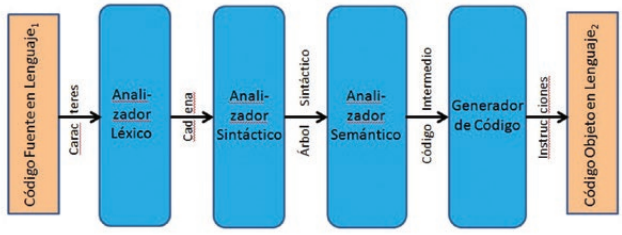


Figura 3. Patrón arquitectónico pipeline

Referencias

[1] L. León. "Los Special Purpose Languages, parte 3". Revista Software Guru #51.
https://sg.com.mx/revista/51/special-purpose-languages-parte-3
[2] L. León. "Los Special Purpose Languages, parte 2". Revista Software Guru #50.
http://sg.com.mx/revista/50/los-special-purpose-languages-2

SG Campus

SGCampus

@SGCampus

www.sgcampus.com.mx

Asegurando la Calidad
de la Experiencia de Usuario

Por Roselyn C. Piñango

Según el ISTQB (*International Software Testing Qualifications Board*), uno de los objetivos de las pruebas, es satisfacer compromisos aplicando la validación de los requisitos de usuario respecto al sistema que está siendo objeto de la prueba. Igualmente este estándar define las pruebas de sistema como aquellas que se realizan desde la perspectiva del usuario y predomina como protagonista el equipo de QA para ejecutarlas [1]. Por otro lado, uno de los atributos no funcionales de la calidad según la norma ISO9126 es la usabilidad que consiste en la capacidad que tiene un software de ser amigable, atractivo y útil al usuario final. Ahora bien, el dinamismo tecnológico ha permitido que usuarios no expertos interactúen con la tecnología como parte de su día a día haciéndolos más exigentes. La validación de requisitos, pruebas de sistema y la usabilidad ya no son suficientes, es necesario que las pruebas incluyan en su alcance la experiencia de usuario (UX).

La experiencia de usuario es el conjunto de factores y elementos relativos a la interacción del usuario con dispositivos o software tecnológico cuyo resultado es la generación de una percepción positiva o negativa de dicho servicio, producto o dispositivo. Ésta depende no solo de los factores relativos al diseño sino además de aspectos relativos a las emociones, sentimientos, confiabilidad y transmisión de la marca (2). El diseño de la experiencia del usuario se define como el proceso de aumentar la satisfacción y lealtad del cliente mediante mejoras en la usabilidad, facilidad de uso y el placer de la interacción entre el cliente y el producto.

El diseño de experiencia de usuario se preocupa por un número mayor de variables que el diseño de interfaz de usuario y no necesariamente están relacionadas con el aspecto visual, por ejemplo: la presentación de los productos, redacción de contenidos, la velocidad de carga, entre otros factores. Algunos de los principios de este tipo de diseño son el color, eficiencia, error humano, estética, íconos, jerarquía visual, legibilidad, mapeo natural y visibilidad, entre otros. Definitivamente esto compromete aún más a los especialistas en ingeniería de requisitos y en sí, al proceso de interacción con los usuarios de manera que el equipo de desarrollo y pruebas pueda conocer las pautas, funcionalidades y expectativas que debe cumplir el sistema.

En el ámbito de las pruebas, esto significa que no será suficiente la base de pruebas que utiliza un equipo de QA (listado de funcionalidades) sino es necesario comprender claramente cuál es la motivación del sistema, qué beneficios traerá, qué se busca con su construcción y qué se espera de él. Para ello es necesario que las pruebas de calidad de la experiencia de usuario cuenten con la participación del usuario, pero no significa que estas pruebas deban realizarse al final del ciclo de vida de desarrollo, al contrario, debe ser una de las primeras exploraciones (por ejemplo por medio de prototipos) a realizar dentro del enfoque de pruebas definido respetando el principio "pruebas tempranas" de ISTQB. Pudiera ser un nuevo nivel de pruebas híbrido probador/usuario.

De de las pruebas más conocidas en este ámbito son las A/B que consisten en dos escenarios controlados que se envían/cargan de forma aleatoria a los usuarios (normalmente una muestra de ellos) con el objetivo de medir la efectividad de un diseño de interfaz y/o contenido en específico. Otro tipo de prueba utilizada es la llamada *think aloud* (piensa en voz alta) en donde se le solicita al usuario que utilice el sistema, narrando sus pensamientos y percepciones del mismo mientras navega por las funcionalidades del sistema. Lo importante es que se contemple desde QA que las pruebas del diseño de experiencia de usuario no es tarea única del profesional de QA, se trata de un trabajo conjunto con el usuario, y aunque algunos autores califican estas pruebas como "costosas", sin duda no es equivalente a desarrollar un sistema que no sea utilizado cuando esté en producción.

CONCLUSIÓN

La tecnología avanza y los usuarios son cada día más exigentes con los sistemas que reciben para su rutina laboral o vida diaria, lo cual significa un reto adicional para los profesionales de QA, en donde el objetivo de satisfacer compromisos lo obliga a ser un tecnólogo multidisciplinario que debe garantizar la mejor experiencia de usuario posible, funcionalidad (y otros atributos de la calidad) bajo la arquitectura idónea para el contexto donde se ejecutará el sistema, convirtiéndolo en un profesional integral.

Referencias

[1] "Foundation Level Syllabus". ISTQB, 2011. http://swqu.ru/rt

Roselyn C. Piñango Díaz es Ingeniero en Computación con especialización en Sistemas de Información. Gerente de QA Factory en Global R, Venezuela. Cuenta con 10 años de experiencia en desarrollo y pruebas de software; y certificaciones: OCE SQL, OCA PL/SQL Developer, CTFL ISTQB, ASE HP ALM v11, CAA SAPB1.

La Capacitación como *Agente de la Transformación*

Por Omar Sánchez

La capacitación práctica y teórica en Tecnologías de Información es de vital importancia, y necesaria a fin de ampliar el desempeño de las actividades de los profesionales del sector, debido a que las organizaciones son cada vez más dependientes de ellas para soportar y mejorar los procesos de negocio que demandan cumplir las necesidades de los clientes y de la propia organización. Esto requiere de una estrategia y transformación bien definida para la mejora del software que las empresas necesitan, por lo que cabe preguntarnos ¿qué valor o aporte le estamos dando al cliente, colaboradores o accionistas, con el uso de este tipo de tecnologías?

Actualmente la capacitación en TI para los profesionales dedicados a la gestión contribuye al desarrollo profesional, y por tanto al económico de las empresas que prestan interés al tema, ya que los servicios de TI conforman la base del modelo de negocio en su totalidad para cualquier tipo de empresa, por ello, las compañías deben encontrar mecanismos que transmitan información relacionada con las actividades de su organización brindando a sus colaboradores conocimientos, habilidades y actitudes que se pretenden para lograr a través de las mejores prácticas en gestión de servicios de TI, un desempeño óptimo.

Lamentablemente muchas organizaciones consideran que la formación profesional de TI es un gasto innecesario, sin darse cuenta que es todo lo contrario, ya que puede ofrecer resultados positivos así como una optimización en la calidad de la infraestructura en el software que maneja la propia organización; es decir, es una inversión que trae beneficios al colaborador, a la organización y, lo más importante, al cliente o usuario final.

Dentro de la industria de TI existen organismos que centran su importancia en el desarrollo de mejores prácticas ITIL (Information Technology Infrastructure Library, por sus siglas en inglés), que se basa en una colección de documentos públicos, fundamentados en procesos y en un marco de optimización en la industria, permitiendo la Gestión de Servicios de TI con calidad y a un costo adecuado, por supuesto que para ello es importante una capacitación en ITIL enfocada a los profesionales involucrados con todos aquellos procesos que se necesitan ejecutar dentro de las organizaciones para la administración y operación de la infraestructura de TI, de tal forma que logre una excelente provisión de servicios a los clientes bajo un esquema de costos congruentes con las estrategias del negocio.

La biblioteca de ITIL se ha convertido en el estándar mundial de capacitación y actualización para la gestión de servicios Informáticos que genera una sinergia entre personas, procesos y tecnología. Tal es el caso de ITSMF (Information Technology Service Management Forum), que anualmente realiza un congreso en la Ciudad de México, sobre dichos temas.

Algunos de los beneficios de una capacitación ITIL para una adecuada gestión del servicio en las tecnologías de información son: maximiza la calidad del servicio apoyando al negocio de forma expresa, ofrece una visión clara de la capacidad del área TI, aumenta la satisfacción en el trabajo mediante una mayor comprensión de las expectativas y capacidades del servicio, minimiza el ciclo de cambios y mejora los resultados de los procesos y proyectos TI, incrementa la rentabilidad del negocio.

Las TI aportan de manera más que sustancial los elementos para lograr las metas y objetivos, cuantitativos como cualitativos, de cualquier tipo de organización o industria, por ello es imposible lograr una gestión exitosa de negocio sin el manejo apropiado de dichas tecnologías. En las organizaciones no solo deben existir mecanismos alineados a los objetivos tácticos como estratégicos, sino también para el resguardo de los activos informáticos, pero sobre todo en capacitación, para saber cómo hacerlo con base a las mejores prácticas normas, guías y estándares.

Omar Sánchez es un consultor especializado en Gestión de Servicios de TI. Actualmente dirige la empresa O2 Systems y es Presidente de ITSMF México, además de ser profesor de asignatura en la Universidad Iberoamericana en el área de Gobierno de TI.

Designing Software Architectures: A Practical Approach

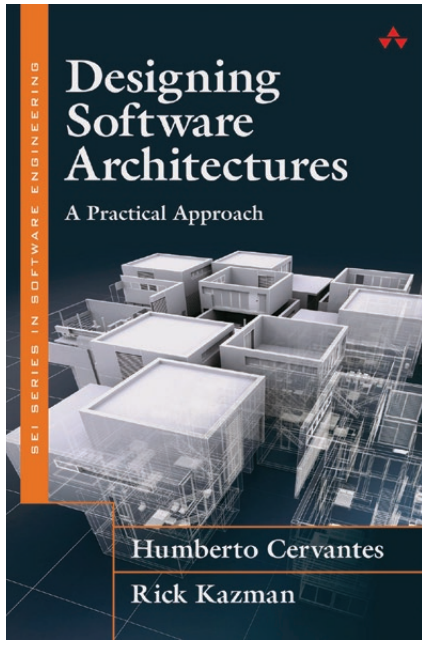
Por Humberto Cervantes (UAM-I)

En esta ocasión tengo el gusto de compartir con ustedes la noticia de la reciente publicación del libro que escribí junto con mi colega Rick Kazman quien es investigador en el Software Engineering Institute (SEI). Nuestro libro lleva por título "Designing Software Architectures - A Practical Approach" (Diseñando Arquitecturas de Software - Un Enfoque Práctico) y fue publicado por la editorial Addison Wesley en la colección "SEI Series in Software Engineering".

Nuestro libro se enfoca en el método de Diseño Guiado por Atributos (Attribute-Driven Design), o ADD; un método de diseño de arquitecturas originalmente creado por el Software Engineering Institute y del cual hablé en la columna "Diseño de la Arquitectura", en SG 29. A pesar de que ADD ha existido por más de una década, se ha escrito relativamente poco al respecto y es difícil encontrar ejemplos de cómo llevarlo a cabo. Esta falta de información ha complicado su adopción y enseñanza. Además, la información que se ha publicado acerca de ADD tiende a ser un tanto abstracta y puede ser difícil conectarla con los conceptos, prácticas y tecnologías que los arquitectos usan en sus actividades cotidianas.

Rick y yo hemos guiado a arquitectos de software durante años en la realización del proceso de diseño y, al mismo tiempo, hemos estado aprendiendo de ellos; por ejemplo, que en la práctica los arquitectos consideran la tecnología de forma temprana en el proceso de diseño y esto es algo que no era parte de la versión original de ADD. Por tal razón, el método original parecía "desconectado" de la realidad para la gente de la industria. En este libro presentamos una versión revisada de ADD en la cual hemos tratado de reducir la brecha entre la teoría y la práctica.

Durante varios años, hemos estado también enseñando arquitectura de software



y la manera en que se realiza el diseño de la misma. Nos hemos dado cuenta que a la gente que no tiene experiencia le es muy difícil diseñar arquitecturas. Este entendimiento nos empujó a crear una "hoja de ruta" que creemos será útil para guiar en la realización del proceso de diseño. También creamos un juego que es útil para enseñar acerca del diseño de arquitecturas, y que se puede considerar como un complemento al libro (ver "Smart Decisions: Un juego para aprender Arquitectura y Big Data" en SG 49).

Nuestro libro está dirigido a cualquier persona que esté interesada en el diseño de arquitecturas de software. Creemos que será particularmente útil para los practicantes que deben realizar esta tarea, pero que actualmente la realizan de forma ad hoc. Los practicantes con experiencia que ya realizan el diseño siguiendo un método establecido también encontrarán nuevas ideas, por ejemplo, cómo dar seguimiento al proceso de diseño usando un tablero Kanban o cómo analizar el diseño usando cuestionarios basados en tácticas. Por último, la gente que está familiarizada con otros métodos del Software Engineering Institute encontrarán información acerca de la manera de combinar ADD con métodos tales como el taller de atributos de calidad QAW, el método de análisis de compromisos arquitectónicos ATAM (ver SG 31) y el método de análisis de costos beneficios CBAM.

Este libro también será útil para estudiantes y profesores de programas de ciencias de la computación o de ingeniería de software. Creemos que los casos de estudio que describimos ayudarán a entender a través de ejemplos la manera de realizar el proceso de diseño más fácilmente. Como dijo Einstein: "El ejemplo no es otra manera de enseñar, es la única manera de enseñar".

Esperamos que nuestro libro permita entender que el diseño puede realizarse siguiendo un método y que el hacerlo de esta manera es de gran ayuda para producir mejores sistemas de software.

Su estructura es de la siguiente manera siguiente:

- **En el capítulo 1** presentamos una introducción breve del concepto de arquitectura de software y del método ADD.

- **En el capítulo 2** discutimos acerca del diseño de arquitecturas de manera más detallada, junto con las entradas principales al proceso de diseño a las cuales llamamos los drivers de la arquitectura. Además, hablamos de los conceptos de diseño que son 'bloques de construcción' a partir de los cuales se construye la arquitectura y que permiten satisfacer estos drivers usando soluciones probadas.

- **El capítulo 3** presenta el método ADD de forma detallada. Presentamos cada uno de

los pasos del método junto con varias técnicas que pueden ser usadas para llevar a cabo estos pasos de manera adecuada.

• **El capítulo 4** presenta un primer caso de estudio que ilustra el desarrollo de un sistema "desde cero". Para este caso de estudio, hemos hecho un esfuerzo por mostrar la manera en que la mayoría de los conceptos descritos en el capítulo 3 son usados en el proceso de diseño, así que este caso de estudio tiene una naturaleza más "académica" (aunque se deriva de un sistema del mundo real).

• **El capítulo 5** presenta un segundo caso de estudio que escribimos junto con arquitectos de software expertos en big data y, por ello, tiene muchos más detalles técnicos que el anterior. Este caso de estudio describe los detalles finos de cómo usar ADD en el diseño de un sistema de big data que involucra diversas tecnologías. Este ejemplo ilustra también lo que consideramos un diseño en un dominio "novedoso", a diferencia del dominio más tradicional usado en el capítulo 4.

• **El capítulo 6** es un caso de estudio más corto que describe el uso a ADD para realizar la extensión de un sistema legado, lo cual es una situación común en la práctica. Este ejemplo demuestra que el diseño arquitectural no es algo que se realice una sola vez, al inicio del desarrollo del sistema, sino que más bien es una actividad que puede ser realizada en distintos momentos del proceso de desarrollo.

• **El capítulo 7** presenta otros métodos de diseño. En nuestra revisión de ADD, adoptamos ideas de otros autores que también han investigado el proceso de diseño y aquí resumimos brevemente sus enfoques como un homenaje a sus propuestas y como una comparativa de ADD a estos métodos.

• **El capítulo 8** discute acerca del tema del análisis de forma detallada. Aún si el tema principal de este libro es el diseño, el análisis se realiza naturalmente como parte del mismo, así que aquí describimos técnicas de análisis que pueden ser usadas durante el proceso de diseño o después de que una parte del diseño ha sido completada. En particular, describimos el uso de cuestionarios basados en tácticas, los cuales son útiles para entender de forma simple y rápida las decisiones que se realizan durante el proceso de diseño.

• **El capítulo 9** describe la manera en que el proceso de diseño embona a nivel organizacional. Por ejemplo, es posible realizar cierta cantidad de diseño en los momentos más tempranos del ciclo de vida con el fin de realizar la estimación del proyecto (ver "Arquitectura y Preventa" en SG 46). Mostramos también cómo ADD puede ser usado con distintos enfoques de desarrollo, como por ejemplo, con las metodologías ágiles o con métodos más establecidos como el Team Software Process (TSP).

• **El capítulo 10** concluye el libro.

Incluímos también dos apéndices. El A presenta un "catálogo de conceptos de diseño" que, como su nombre indica, es un catálogo de distintos tipos de conceptos de diseño que pueden ser usados para un dominio aplicativo particular (ver "Conceptos de Diseño: Patrones, Tácticas y Frameworks" en SG 38). Este catálogo presenta conceptos de diseño que recopilamos de distintas fuentes y refleja la manera en que los arquitectos con experiencia trabajan en el mundo real. En este caso, el catálogo contiene una muestra de conceptos de diseño usados en el caso de estudio presentado en el capítulo 4. El apéndice B provee un conjunto de cuestionarios basados en tácticas para los siete atributos de calidad más comunes, así como un cuestionario adicional para DevOps.

CONCLUSIÓN

Debo decir que me llena de orgullo el haber escrito un libro para la prestigiosa colección editorial del Software Engineering Institute que, además, es el nuevo material de referencia para el curso "Software Architecture Design and Analysis" que se imparte en el SEI. Espero que sea de mucha utilidad para la comunidad de desarrollo en México y en el resto del mundo.

Quiero aprovechar para agradecer a Pedro Galván y su equipo por la oportunidad que me han dado a lo largo de estos años de escribir esta columna de arquitectura de software. Como se puede apreciar en las distintas referencias que hice a números pasados de la revista, muchos de los temas desarrollados de forma detallada en el libro han sido presentados de forma resumida a lo largo de los años en esta columna.

Para terminar, incluyo una liga en donde se puede adquirir el libro:
https://goo.gl/Wd01TL

El Dr. Humberto Cervantes es profesor-investigador en la UAM-Iztapalapa. Además de realizar docencia e investigación dentro de la academia en temas relacionados con arquitectura de software, realiza consultoría y tiene experiencia en la implantación de métodos de arquitectura dentro de la industria. Ha recibido diversos cursos de especialización en el tema de arquitectura de software en el SEI, y está certificado como ATAM Evaluator y Software Architecture Professional por parte del mismo. (www.humbertocervantes.net)

Contenedores

UN POCO DE HISTORIA

—

Por Gunnar Wolf

Gunnar Wolf es administrador de sistemas para el Instituto de Investigaciones Económicas de la UNAM y desarrollador del proyecto Debian GNU/Linux.
http://gwolf.org

Recuerdo la primera vez que tuve contacto con la virtualización. En 2005, en un congreso, participé en un breve taller en el que nos presentaron esta (entonces) característica tan única de las arquitecturas IBM de gama alta: los servidores s390.

¡Parecía magia! Varios sistemas operativos corriendo a velocidad nativa, compartiendo armónicamente una misma computadora. Un hipervisor gestionando y mostrando estadísticas de uso en tiempo real. Claro, a esas alturas yo ya conocía el mecanismo que empleaba VMWare para virtualizar sobre la arquitectura PC desde fines de los noventa, pero la diferencia en el rendimiento y la confiabilidad era muy notoria. Por esos años, relativamente muy poca gente consideraba la oferta de VMWare para entornos de producción.

Un año más tarde, el panorama cambió radicalmente. Tanto Intel como AMD comenzaron a incorporar extensiones de virtualización en sus procesadores de arquitectura x86 —las instrucciones y niveles de protección necesarios para permitir este mágico aislamiento que hasta entonces estaba reservado a los sistemas de muy alto costo. Si bien la virtualización apareció primero en la gama alta de estas compañías, hoy en día (diez años más tarde) prácticamente todos sus procesadores las incluyen. Y sí, cambiaron fuertemente la forma de trabajo en los centros de datos, las ofertas de servicio de los proveedores de infraestructura— y permitieron el nacimiento del, permítanme esta redundante cacofonía, nebuloso término de "la nube".

Por un par de años, hubo un claro surgimiento de la "virtualización basada en hardware". Proveedores de servicio de todo tamaño comenzaron a ofrecer "servidores privados virtuales" a precios que antes nunca hubiéramos esperado. Los administradores de sistemas de todo el mundo comenzamos a cambiar nuestra mecánica de trabajo, "particionando" el trabajo que realizaban nuestros grandes servidores monolíticos en cargas especializadas, a ser repartidas en cuantos servidores virtuales fuera necesario. No tardaron en aparecer numerosas tecnologías base, libres y propietarias.

DE VIRTUALIZACIÓN A PARAVIRTUALIZACIÓN

Quedaba aún mucho por mejorar, sin embargo, si bien la ejecución de código dentro de las máquinas virtuales ocurría ya a velocidad nativa, había inocultables capas de emulación que era necesario cruzar. ¿A qué me refiero? Una máquina virtual aparenta ser una computadora completa, al lanzar una, vemos cómo una interfaz comparable con el BIOS lanza al gestor de arranque, el sistema operativo se inicializa... la máquina virtual cree tener las tarjetas de red, unidades de disco y demás periféricos que le indiquemos (que normalmente corresponden con modelos simples y antiguos). El manejo del video es particularmente curioso; lo más común es que se presente como una tarjeta de video tonta, sin aceleración 3D, que interfacea mediante un *framebuffer*. Todos estos dispositivos tienen que ser emulados, y toda emulación tiene su costo.

Hoy en día, prácticamente la totalidad de herramientas de virtualización ofrece más bien un híbrido, entrando al terreno de la paravirtualización. ¿Qué es? Es cuando el engaño no es completo, cuando el hardware virtual ofrecido al sistema operativo ya no pretende ser una computadora completa, sino que el sistema operativo huésped "sabe" que está siendo virtualizado y coopera en la tarea. Esta cooperación estiba en que, en vez de buscar controlar a su hardware como si fuera real, lo toma como un buffer glorificado y envía, más bien, solicitudes al anfitrión (host) para cumplir lo que los programas requieren. Una diferencia aparentemente sutil, pero que —sobre todo en los casos de cargas de trabajo orientadas a la entrada y salida, como las bases de datos y el manejo de red— típicamente llevan a una diferencia de rendimiento de diez veces tanto.

Virtualización y paravirtualización brindan grandísimas ventajas a los administradores, y una en la que insistiré en este punto es que el "engaño" es a nivel sistema entero, cada computadora virtual puede ejecutar un sistema operativo distinto completo. Esto es particularmente útil al hablar de proveedores de servicio en la nube, pues cada cliente tendrá la flexibilidad absoluta de correr en su computadora el software que desee. Sin embargo, hacia adentro de las organizaciones, sigue siendo subóptimo.

RAÍCES, JAULAS, CONTENEDORES

Como siempre, la respuesta a una nueva necesidad viene del pasado, con nuevas maneras de emplear lo ya conocido, de refinamientos técnicos que mejoran

la implementación de la idea —y en este caso, de la mano de bastante mercadotecnia que llevó a mucha gente a ver hacia el lugar obvio.

Los contenedores pueden definirse como virtualización a nivel sistema operativo. Los usuarios de máquinas virtuales ya no serán engañados con lo que parece ser una computadora completa, sino que únicamente con algo que dice ser un sistema operativo dedicado a ellos. Y, siendo el sistema operativo el principal encargado de agrupar al hardware en clases, gestionar su manipulación de bajo nivel, y proveer abstracciones uniformes para los programas que corren en espacio de usuario... la diferencia es enorme.

Pero vamos por partes, ¿cómo se originaron los contenedores?, ¿cuáles son las ideas primigenias?

En 1982, Bill Joy, uno de los arquitectos del Unix de BSD y fundador de Sun Microsystems, creó la llamada al sistema chroot() para auxiliarse en el desarrollo de Unix. Dado que desarrollaba en la misma computadora donde hacía sus pruebas, quería poder mantener aislado su entorno de desarrollo y construcción del entorno de pruebas. La manera más sencilla (y, además, elegante) de hacerlo es por medio de esta llamada, se invoca con un argumento, y a partir del momento en que un proceso la llama con el nombre de un directorio y hasta el final de su ejecución, el sistema operativo le dará a dicho proceso una vista parcial del sistema, limitada al directorio que le es indicado. Esto es, tras un chroot('/home/gwolf/test'), si el proceso en cuestión da un cd /, el sistema operativo lo llevará a /home/gwolf/test.

En los manuales de sistema es común encontrar advertencias de seguridad — chroot() no está pensado para proveer aislamiento, únicamente conveniencia — Hay varias maneras por medio de las cuales un proceso puede escapar de esta jaula, además de que su funcionamiento se limita a restringir la vista del sistema de archivos, sin proteger otros aspectos del sistema como las interfaces de red, la lista de procesos, o el acceso a dispositivos.

El primer sistema operativo en ofrecer lo que hoy conocemos como contenedores (bajo el nombre de jails) fue la versión 4 de FreeBSD, liberada en el año 2000. Poco después, esta funcionalidad apareció en Solaris, Linux e incluso para Windows; en los dos últimos casos, esto fue inicialmente mediante productos de terceros (*Parallels Virtuozzo Containers para Windows, Vserver y OpenVZ para Linux, OpenVZ* siendo una rama libre de *Virtuozzo*). Varios años después de la introducción de esta funcionalidad por medio de terceros, tanto en el caso de Linux (con lxc desde 2009) como recientemente en el de Windows (desde 2015) han sido incorporados a la plataforma base del sistema operativo.

La virtualización a nivel sistema operativo implica que los contenedores engañarán a los procesos huéspedes presentándoles una vista limitada en los siguientes aspectos:

- Tablas de procesos
- Señales y comunicación entre procesos (IPC)
- Interfaces de red
- Dispositivos de hardware
- Límites en el consumo de recursos (espacio en disco, memoria, o ciclos de CPU)
- Información del sistema, como el nombre del equipo o la hora actual

Esto permite a los árboles de procesos que habitan dentro de un contenedor tener una vista de sistema completo, viéndose obligados a respetar las asignaciones de recursos de los demás contenedores. Uno de los principales atractivos para los administradores de sistemas es que un contenedor cuyos servicios no están siendo activamente utilizados permanece efectivamente dormido, con un nivel de consumo de recursos verdaderamente mínimo — A diferencia de la virtualización de sistema entero, en este caso el sistema operativo conoce directamente todos los eventos a que están esperando cada uno de los procesos, y mientras dicho evento no ocurra, los contenedores no pasarán de ser una estructura en memoria.

A últimos años, los contenedores se han ido popularizando ya no únicamente como una herramienta para que el administrador de sistemas particione lógicamente la carga de trabajo de servidores o provea infraestructura de cómputo con acceso privilegiado a distintos usuarios manteniendo garantías de seguridad, sino también como mecanismo de distribución de software, instalado y preconfigurado —la implementación más conocida es el proyecto *docker*. En lo particular, como administrador de sistemas de vieja guardia y formado en la escuela tradicionalista, le veo importantes carencias a su planteamiento... pero sin duda ha logrado una importante penetración de mercado.

Como sea, la tecnología de contenedores es una genial herramienta con la que contamos hoy, que ha cambiado de raíz la manera en que se despliegan los servicios. Invito a todos los lectores que no se hayan aún adentrado a comprenderla y utilizarla, a no dudar en hacerlo.

1

INTEL JOULE

Intel presenta su más reciente oferta para makers, el módulo Intel Joule™. Este módulo de cómputo destaca por su poder, ya que cuenta con un procesador quad-core de 64 bits a 1.7 GHz, 4 GB de RAM, 16 GB de almacenamiento interno (eMMC), Intel HD Graphics, interfaces USB 3.0, Bluetooth LE, y WiFi de banda doble. Estas características permiten que Joule pueda utilizarse en tareas que requieren gran poder de cómputo y que no son viables para módulos anteriores de Intel como Edison o Galileo. Una de los escenarios de uso para Joule es el de visión computacional utilizando la tecnología RealSense. Otra posibilidad es que pueda procesar algoritmos de machine learning localmente. Además de su poder, el Intel Joule también hace gala de gran versatilidad. Puede arrancar (bootear) desde almacenamiento interno, tarjetas de memoria microSD, y discos externos vía USB. Por default utiliza una distribución Linux optimizada para IoT, pero es compatible con otros sistemas operativos como Windows 10 IoT Core y Ubuntu Core.

2

KANO KITS

Kano, la empresa que hace kits de construcción+programación dirigidos a niñas y niños, lanzó tres nuevos kits: una cámara, una bocina y un tablero de pixeles. En el espíritu DIY (hágalo usted mismo), todo viene desarmado para que puedas ensamblarlo y entender cómo interactúan los distintos componentes. Pero el verdadero DIY está del lado del software, ya que el comportamiento de los dispositivos es personalizable y programable. Los dispositivos cuentan con conectividad WiFi e incluyen sensores con conectividad USB por lo que puedes detectar gestos y sonidos, y reaccionar en base a eventos. Por ejemplo, qué pasa cuando sacudes la bocina, o cuando detecta que alguien se acerca. En el caso de la cámara, incluso puedes crear tus propios filtros. Todo esto se programa de forma visual utilizando el ambiente de desarrollo de Kano basado en bloques (similar a Scratch).

https://kano.me

3

GOPRO KARMA VS. DJI MAVIC PRO

GOPRO KARMA

GoPro decidió incursionar en el espacio de los drones y anunció el lanzamiento de Karma, un dron cuadricóptero que se puede doblar para hacerlo portátil. Viene listo para que le montes una cámara GoPro (4 o 5) e incluye un dispositivo de estabilización que puedes utilizar con ella aunque no esté montada en el Karma.

Todo parecía indicar que Karma sería un éxito, pero apenas unos días después, la empresa china DJI anunció el lanzamiento de su nuevo dron, el Mavic Pro. Y parece ser que todo lo que puede hacer el Karma, el Mavic Pro puede hacerlo igual o mejor. Porque también es flexible (más ligero y compacto que el Karma), es más rápido (65 km/h vs. 56 km/h), tiene mayor rango (7 km. vs. 3 km.), puede volar más tiempo (27 min. vs. 20 min.) y tiene control automático de vuelo. El Karma lleva ventaja en cuestión de flexibilidad con la cámara, ya que mientras en el Mavic la cámara ya viene incluida y está empotrada en el dron, en el Karma puedes desmontarla para usarla por separado. Hasta el momento no hay comparativos de la calidad de imagen de ambas cámaras, pero suponemos que dada su experiencia en este espacio, el Karma debería salir mejor librado.

Solo el tiempo dirá cual de estos drones resulta vencedor, pero la verdad es que con tener cualquiera de los dos estaríamos bastante satisfechos.

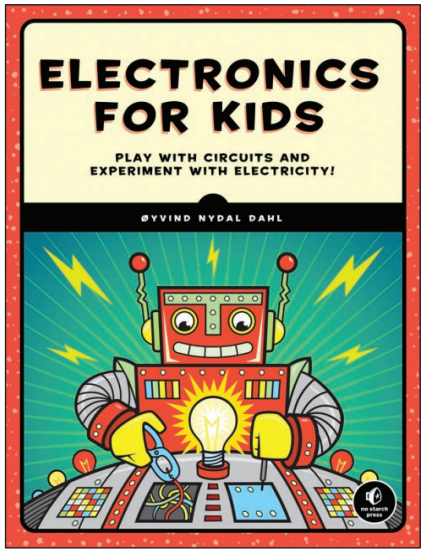
DJI MAVIC PRO

4

STAR WARS FORCE BAND

Sabemos que esto es un juguete, pero también sabemos que si la fuerza está contigo será más divertido si la canalizas a través de la Star Wars™ Force Band™. Una pulsera que detecta gestos (movimientos) y los transmite como comandos al BB-8 (hecho por la misma empresa), que te permite también controlar al droide de juguete. La Force Band también incluye otras modalidades de entrenamiento sin necesidad del BB-8, tales como combate con espadas, y un juego de búsqueda de holocrons. Lo más interesante sería que la Force Band se pudiera conectar con otros dispositivos para poder controlarlos por medio de gestos. Por el momento, esa capacidad no está disponible, pero cruzamos los dedos para que Sphero pronto libere dicha capacidad. Y si no, confiamos en el poder de la ingeniería inversa.

Humor



ELECTRONICS FOR KIDS
Øyvind Nydal Dahl. No Starch Press, 2016

1

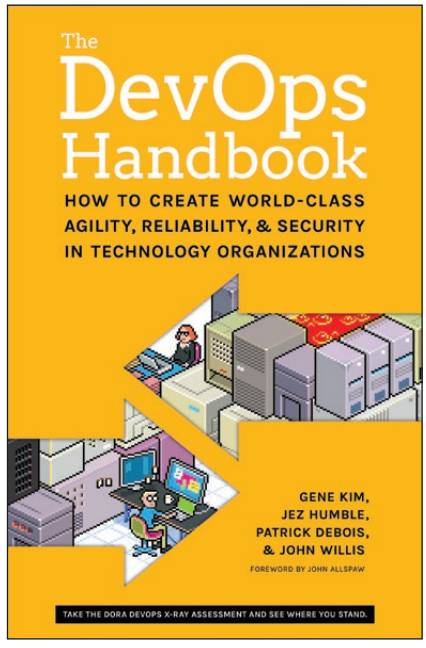
Nuestros amigos de No Starch Press continuan con su tradición de publicar libros sobre ciencia y tecnología dirigidos a niños y ahora nos presentan "Electronics for Kids" por Øyvind Nydal Dahl, un libro dedicado a explicar aspectos fundamentales de electrónica por medio de una colección de proyectos prácticos.

El libro está compuesto por 23 proyectos divididos en tres secciones. La primera sección consiste de proyectos básicos orientados a entender la electricidad: se construye una batería con un limón, un magneto eléctrico con un tornillo y un generador, usando imanes y un vaso. Los proyectos de la segunda sección se enfocan en circuitos y se usan componentes tales como LEDs, resistencias, capacitores, relevadores y sensores. La tercera y última sección está dedicada a proyectos de electrónica digital usando compuertas lógicas y memoria, e incluye un proyecto final para construir un juego aplicando los conceptos aprendidos a lo largo de todo el libro.

Electronics for Kids cubre todo lo que podemos pedir de un libro educativo en ciencia y tecnología para niños: enseña conceptos de forma entretenida, tiene casos prácticos, e incluso está bonito.

THE DEVOPS HANDBOOK: HOW TO CREATE WORLD-CLASS AGILITY, RELIABILITY, AND SECURITY IN TECHNOLOGY ORGANIZATIONS
Por Gene Kim, Patrick Debois, John Willis, Jez Humble, John Allspaw. IT Revolution Press, 2016

2



Gene Kim, autor principal de "The Phoenix Project" (IT Revolution Press, 2014) —posiblemente el libro más recomendado en TI en los últimos años ya que ejemplifica de forma clara y amena los retos que enfrentan los líderes de organizaciones de TI para entregar software de manera ágil y confiable—, regresa ahora con un equipo de súper-estrellas del DevOps para brindarnos esta guía a profundidad sobre el tema.

The DevOps Handbook muestra cómo establecer la cultura y prácticas necesarias para maximizar el aprendizaje organizacional y aumentar la satisfacción tanto de clientes como empleados. Los autores comparten una guía práctica de cómo aplicar los conceptos de DevOps en el mundo real, integrando disciplinas de desarrollo de software, aseguramiento de calidad, operación de TI, seguridad informática y gestión de productos. Todo esto se ejemplifica por medio de casos de estudio de empresas como Google, Capital One, Target y Netflix, entre otras.

The DevOps Handbook es una lectura obligada para cualquiera que busque entender, explicar y adoptar DevOps, tanto su cultura como sus procesos y herramientas.

Las personas que compran la versión impresa de este libro reciben un código que pueden utilizar para obtener un diagnóstico DevOps personalizado usando la metodología DORA, creada por los autores del libro. Así que si eso te interesa, considera comprar la versión en papel.