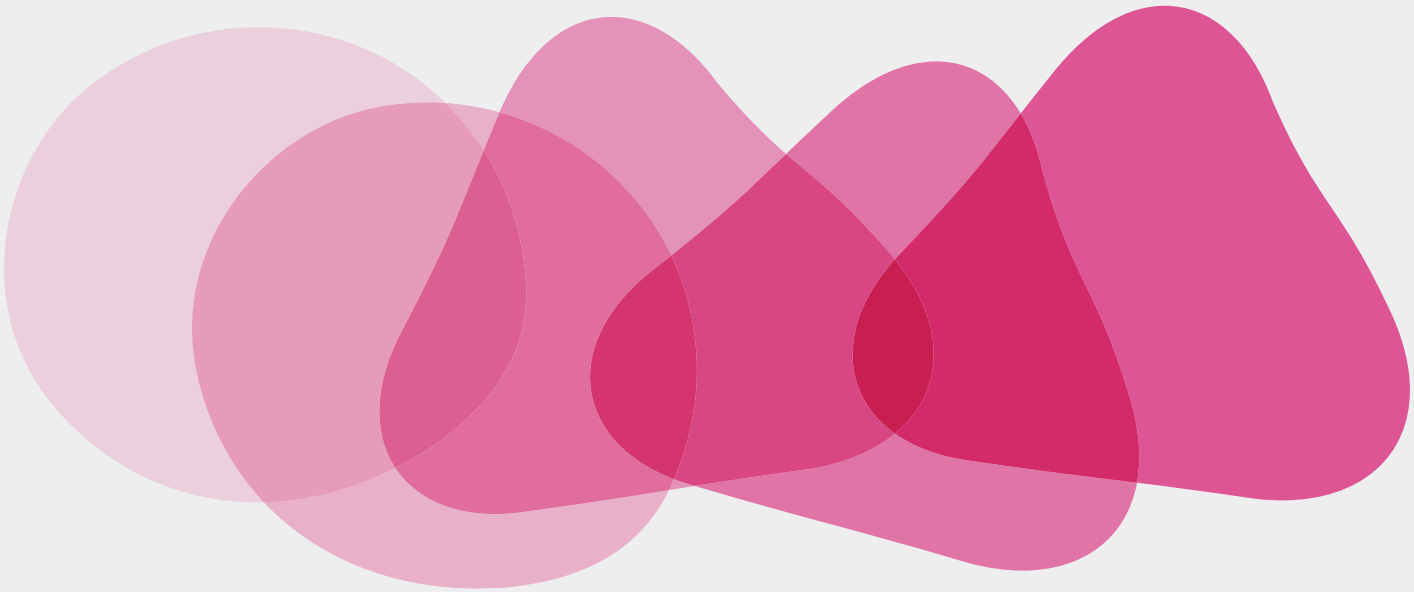
ThoughtWorks®

# TECHNOLOGY RADAR *VOL.19*

Nuestro pensamiento sobre tecnología y las tendencias que dan forma al futuro



*thoughtworks.com/es/radar*

***#TWTechRadar***

# CONTRIBUYENTES

*El Radar Tecnológico está preparado por la Junta Asesora de Tecnología de ThoughtWorks, compuesta por:*


Rebecca Parsons (CTO) | Martin Fowler (Chief Scientist) | Bharani Subramaniam | Camilla Crispim | Erik Doernenburg | Evan Bottcher | Fausto de la Torre | Hao Xu | Ian Cartwright | James Lewis | Jonny LeRoy | Ketan Padegaonkar | Lakshminarasimhan Sudarshan | Marco Valtas | Mike Mason | Neal Ford | Ni Wang | Rachel Laycock | Scott Shaw | Shangqi Liu | Zhamak Dehghani

Traducido por: Ana Tellería, Alexander Castillo, Alexandra Granda, Carlos Oquendo, Daniel Santibañez, Daniela Vásquez, Daniela Mora, David Martín, David Montaño, David Román, David Corrales, Eduard Maura, Ferrán Gomis, Gonzalo Martínez, Gorka López de la Torre, Ignacio Morales, Iván Ortega, Javier Iglesias, Jennifer Carrillo, Jesús Cardenal, Jonathan Morocho, José Puebla, Juan López, Lorena Campos, Luis García, Luisa Emme, María José Lalama, Mateo Rojas, Miguel Hernandez, Miquel Álvarez, Nathaly Montoya, Nicolas Justiniano, Nilet Soto, Omar Agudo, Reynier Pupo, Roberto Almeida, Tania León, Tex Albuja, Victoria Valverde y Victor Ferrás.

Esta edición del Radar Tecnológico de ThoughtWorks se basa en un encuentro de la Junta Asesora de Tecnología, que se reunió en Atlanta en octubre de 2018.

# ¿QUÉ HAY DE NUEVO?

*Temas destacados en esta edición:*

## NUBES PEGAJOSAS

Proveedores de la nube saben que compiten en un mercado estrecho y que para tener éxito necesitan registrar y retener a clientes por largos plazos. Entonces, para ser competitivos, los proveedores se apuran para añadir nuevas características, y hemos observado que han llegado a la paridad de características, lo que se refleja en que hayamos colocado a AWS, GCP, y Azure en el anillo de PROBAR. Sin embargo, una vez que los clientes estén registrados, estos proveedores tienden a crear una conexión tan pegajosa con sus clientes para desalentar la migración a otros proveedores. Frecuentemente esto se manifiesta en una fuerte dependencia en la suite específica de servicios y herramientas, ofreciendo una mejor experiencia de desarrollo mientras los clientes se queden con ellos. Algunas compañías se sorprenden cuando esta dependencia se vuelve aparente, a menudo cuando la decisión de mover partes o toda su carga de trabajo hacia otra nube o al encontrar sus usos y pagos en la nube están fuera de control. Alentamos a nuestros cliente usar o la técnica de correr costo como función fitness de arquitectura para monitorear el costo de la operación como un indicador de pegajosidad o Kubernetes y contenedores para incrementar la portabilidad de carga de trabajo y reducir el costo del cambio a otra nube mediante infraestructura como código. En este Radar, también introducimos dos nuevas herramientas de automatización para la nube, Terragrunt y Pulumi. Aunque alentamos el evaluar las nuevas ofertas de tu proveedor de la nube a través del lente de la pegajosidad, advertimos en contra del uso genérico de la nube. En nuestra experiencia, los gastos generales de crear y mantener capas de abstracción que sean nube agnósticas sobrepasan el costo de salida de un proveedor particular.

## ANTI-PATRONES EMPRESARIALES PERSISTENTES

Sin importar lo rápido que la tecnología cambie, las empresas todavía encuentran maneras de re-implementar anti-patrones del pasado. Muchas de nuestras entradas en resistir denuncian a un viejo lobo que se esconde con nuevas ropas de oveja: comportamiento Bus de Servicio Empresarial (enterprise service bus - ESB por sus siglas en Inglés) implementado en plataformas de transmisión de eventos—Recreando anti-patrones ESB con Kafka, Arquitectura de Microservicios en Capas, Paquetes Hambrientos de Datos, Portales de API Demasiado Ambiciosos, Plataformas de Bajo Código y otras antiguas prácticas nocivas. El problema fundamental, como siempre, es el equilibrio entre aislamiento y acoplamiento: aislamos cosas para hacerlas manejables desde una perspectiva técnica, pero después necesitamos añadir coordinación para utilizarlas al resolver problemas de negocio, resultando en alguna forma de acoplamiento. Así, estos viejos anti-patrones siguen re-emergiendo. Nuevas arquitecturas y herramientas proveen maneras apropiadas de resolver estos problemas, pero eso requiere esfuerzo deliberado para entender cómo usarlas apropiadamente, y no simplemente caer en re-implementar patrones antiguos con tecnología nueva y brillosa.

## PRÁCTICAS DURADERAS DE INGENIERÍA

Un efecto secundario de la mayor rapidez con la que se produce la innovación tecnológica es un patrón repetitivo de expansión y contracción. Cuando aparece una innovación que cambia la forma en que pensamos sobre algunos aspectos del desarrollo de software, la industria se apresura en adoptarla: uso de contenedores, reactivos *fronted*, *machine learning*, etc. Esa es la fase de expansión. Pero, para hacer que estos "nuevos elementos" sean realmente efectivos hay que darse cuenta cómo aplicarles prácticas de ingeniería duraderas: entrega contínua, pruebas, colaboración y demás. La fase de contracción ocurre cuando nos damos cuenta cómo usamos estas nuevas capacidades de manera efectiva, creando bases firmes que posibiliten la siguiente explosión expansiva. Durante esta fase aprendemos cómo aplicar prácticas como las pruebas automáticas exhaustivas, y la automatización de secuencias de pasos recurrentes dentro del contexto de la nueva tecnología. Con frecuencia, esto va de la mano con la creación de nuevas herramientas de desarrollo. Aún cuando parece que la introducción de innovaciones tecnológicas por sí solas generan avances en nuestra industria, es la combinación de la innovación con prácticas duraderas de ingeniería lo que apuntala el progreso continuo.

## RITMO = DISTANCIA / TIEMPO

Nuestros temas usualmente resaltan patrones que notamos en varias de las entradas del Radar actual, pero éste hace referencia a todas las entradas publicadas en todas las ediciones del Radar. Nos dimos cuenta (e hicimos una investigación para soportar nuestras afirmaciones) que la cantidad de tiempo que los *blips* permanecen en el Radar está reduciéndose. Cuando empezamos con el Radar, hace una década, lo usual era que las entradas permanezcan unas dos ediciones en él (aproximadamente un año) sin movimiento antes que se desvanezcan de manera automática. Sin embargo, como se indica por la fórmula en el título de este tema*, ritmo = distancia / tiempo*, el cambio en el ecosistema de desarrollo de software continúa acelerándose. El tiempo ha permanecido constante (seguimos publicando el Radar dos veces al año), pero la distancia recorrida en términos de innovación tecnológica se ha incrementado de forma notoria, dando aún más evidencia de lo que es obvio para cualquier observador astuto: el ritmo de cambio tecnológico sigue incrementándose. Vemos este ritmo acelerado en todos los cuadrantes del Radar y también en el apetito de nuestros clientes para adoptar nuevas y diversas opciones tecnológicas. En consecuencia, modificamos nuestro criterio base para esta edición del Radar: ahora, cada entrada debe aparecer en el Radar por sus méritos actuales, es decir, ya no dejaremos que las entradas permanezcan por defecto. Hemos hecho este cambio luego de cuidadosas deliberaciones, al sentir que nos permite capturar de mejor manera el ritmo frenético de cambio siempre presente en el ecosistema tecnológico.

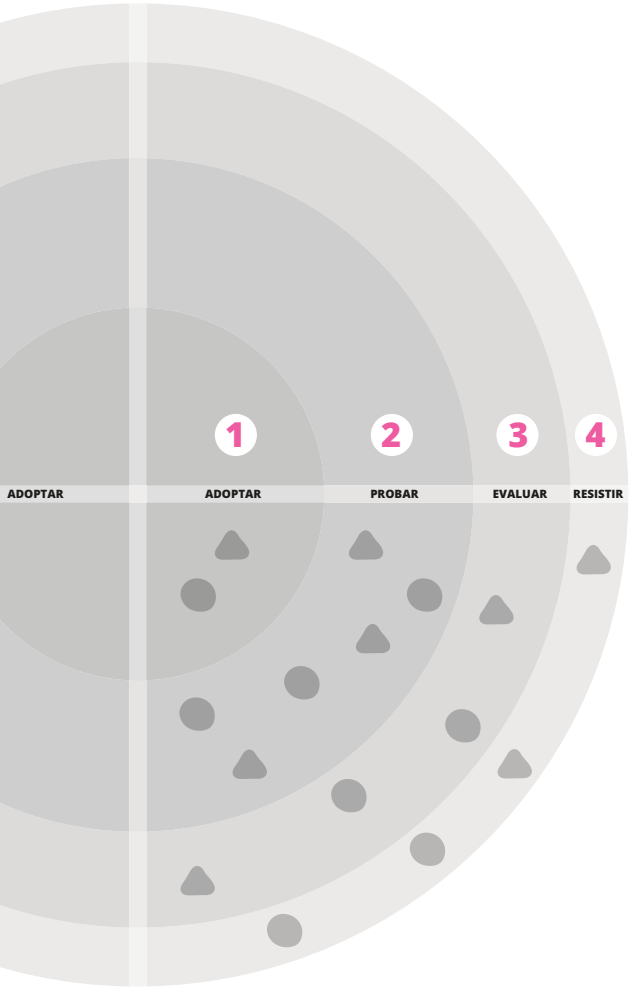
# SOBRE EL RADAR

Los Thoughtworkers somos apasionados por la tecnología. La construimos, investigamos, probamos, liberamos su código fuente, escribimos sobre ella y constantemente queremos mejorarla para todas las personas. Nuestra misión es liderar la excelencia tecnológica y revolucionar las TI. En soporte de esta misión, creamos y compartimos el Radar Tecnológico de ThoughtWorks. La Junta Asesora de Tecnología de ThoughtWorks, un grupo de líderes senior en la tecnología dentro de ThoughtWorks, es quien crea el Radar. Se reunen regularmente para discutir la estrategia tecnológica global de ThoughtWorks y las tendencias tecnológicas que impactan significantemente nuestra industria.

El Radar captura los resultados de las discusiones de la Junta Asesora de Tecnología en un formato que provee valor a un amplio rango de personas interesadas, desde gente desarrolladora hasta CTOs. El contenido pretende ser un resumen consiso.

Los alentamos a que exploren estas tecnologías para más detalle. El Radar es gráfico por naturaleza, agrupando items en técnicas, herramientas, plataformas y lenguajes & frameworks. Cuando los items del Radar pueden aparecer en múltiples cuadrantes, elegimos el que parezca más apropiado. Después agrupamos estos items en cuatro anillos para reflejar nuestra opinión actual sobre ellos.

*Para mayor información sobre el Radar, navega en thoughtworks.com/es/radar/faq*




## UNA MIRADA AL RADAR

**1 ADOPTAR**

Estamos convencidos de que la industria debería adoptar estos ítems. Nosotros los utilizamos cuando es apropiado para nuestros proyectos.

**2 PROBAR**

Vale la pena probarlos. Es importante entender cómo desarrollar estas capacidades. Las empresas deberían probar esta tecnología en proyectos en que se puede manejar el riesgo.

**3 EVALUAR**

Vale la pena explorar, con la comprensión de cómo podría afectar a su empresa.

**4 RESISTIR**

Proceder con precaución.

**NUEVO O MODIFICADO**

**NINGÚN CAMBIO**

Los items que son nuevos o han tenido cambios significativos desde el último Radar son representados por triángulos, mientras que los items que no han cambiado son representados por círculos.

Nuestro Radar tiene una visión hacia el futuro. Para hacer espacio a nuevos items, hemos retirado los items que no han sufrido cambios recientes, lo cual no es un reflejo de su valor sino más bien del espacio limitado disponible en nuestro Radar.

# EL RADAR

## TÉCNICAS

**ADOPTAR**
1. Event Storming

**PROBAR**
2. 1% canary *NUEVO*
3. Bounded Buy *NUEVO*
4. Crypto shredding *NUEVO*
5. Cuatro métricas clave *NUEVO*
6. Multi-account cloud setup *NUEVO*
7. Observabilidad como código *NUEVO*
8. Estrategia de proporción de riesgo del vendedor *NUEVO*
9. Coste de ejecución como función de aptitud de arquitectura *NUEVO*
10. Secretos como servicio *NUEVO*
11. Security Chaos Engineering
12. Datos de versiones para analíticas reproducibles *NUEVO*

**EVALUAR**
13. Katas de Caos *NUEVO*
14. Distroless Docker images *NUEVO*
15. Entrega incremental con costos *NUEVO*
16. Escaner de Configuración de Infraestructura
17. Verificación de compilación descendente antes del commit *NUEVO*
18. Malla de Servicios

**RESISTIR**
19. Manipulación de clusteres Hadoop con estado usando herramientas de gestión de configuración *NUEVO*
20. Uso genérico de la nube
21. Arquitectura de servicios en capas *NUEVO*
22. Gestión de datos maestros *NUEVO*
23. Envidia de Microservicios
24. Eventos petición-respuesta en flujos de interacción con el usuario *NUEVO*
25. RPA *NUEVO*

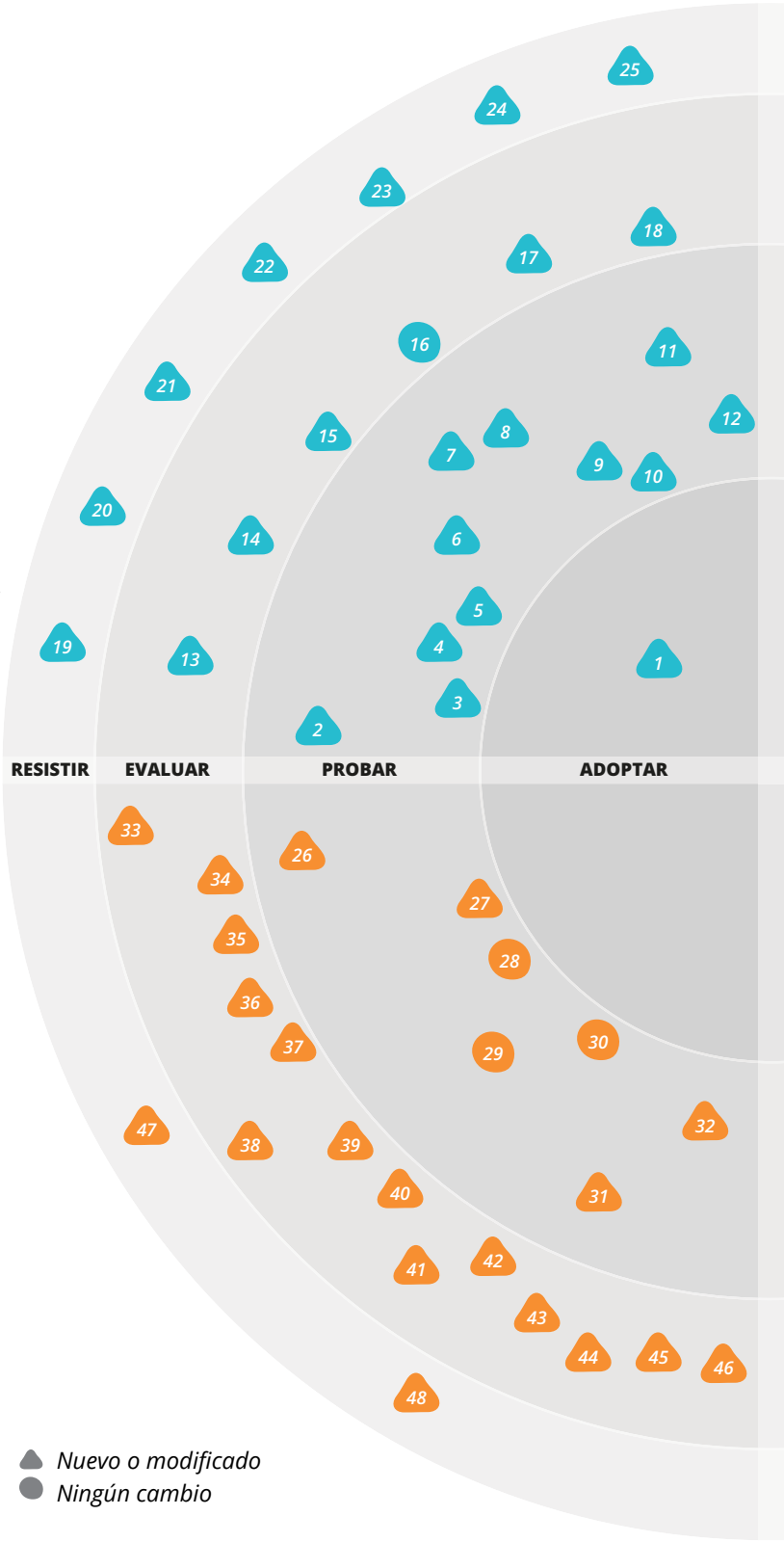
## PLATAFORMAS

**ADOPTAR**

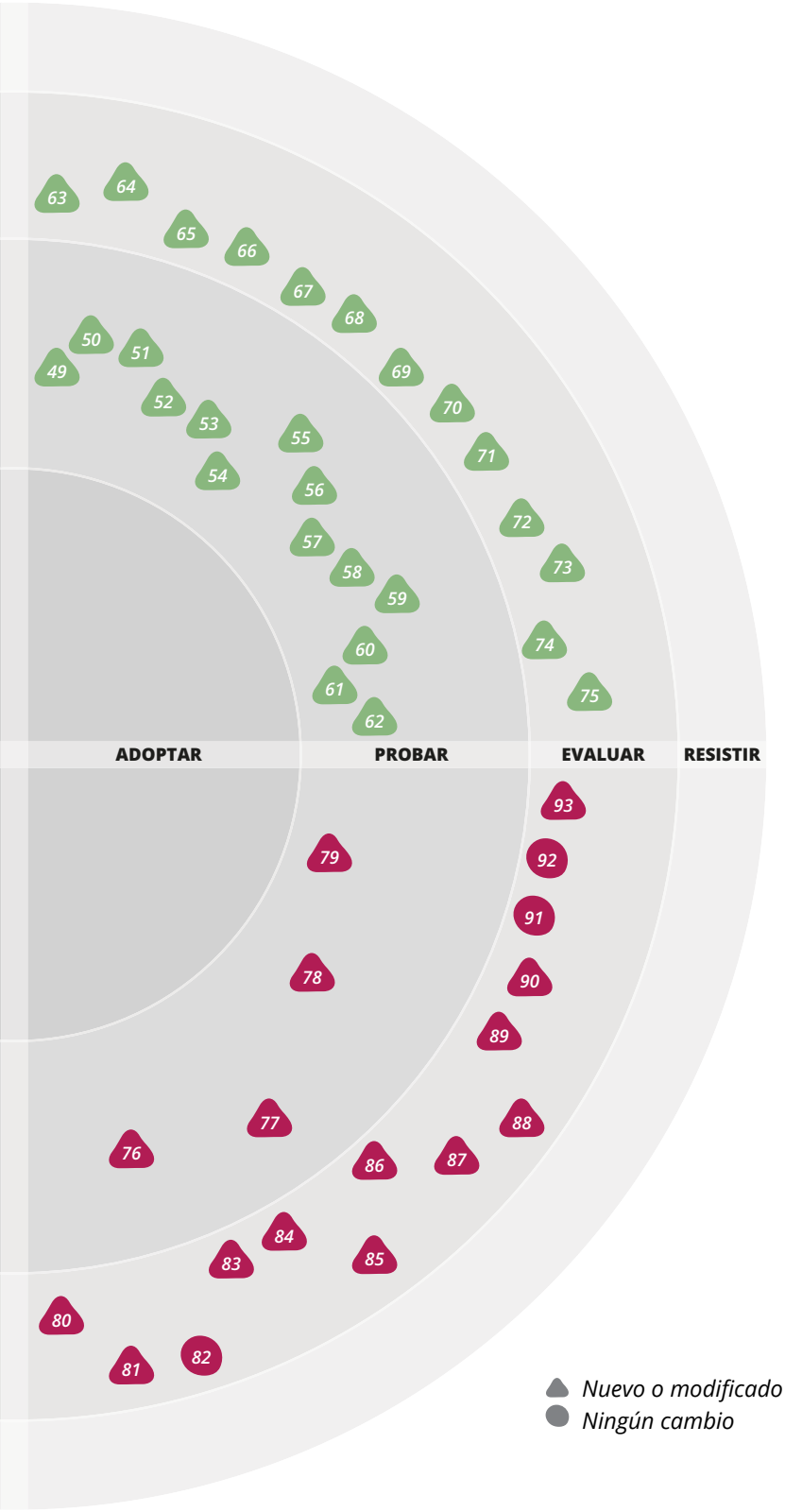
**PROBAR**
26. Apache Atlas *NUEVO*
27. AWS
28. Azure
29. Contentful
30. Google Cloud Platform
31. VPC Compartida *NUEVO*
32. TICK Stack

**EVALUAR**
33. Azure DevOps *NUEVO*
34. CockroachDB *NUEVO*
35. Debezium *NUEVO*
36. Glitch *NUEVO*
37. Google Cloud Dataflow *NUEVO*
38. gVisor *NUEVO*
39. IPFS *NUEVO*
40. Istio *NUEVO*
41. Knative *NUEVO*
42. Pulumi *NUEVO*
43. Quorum *NUEVO*
44. Resin.io *NUEVO*
45. Rook *NUEVO*
46. SPIFFE *NUEVO*

**RESISTIR**
47. Paquetes Data-Hungry *NUEVO*
48. Plataformas de código bajo *NUEVO*



# EL RADAR




## HERRAMIENTAS

**ADOPTAR**

**PROBAR**
49. acs-engine *NUEVO*
50. Archery *NUEVO*
51. ArchUnit
52. Cypress
53. git-secrets *NUEVO*
54. Headless Firefox
55. LocalStack *NUEVO*
56. Mermaid *NUEVO*
57. Prettier *NUEVO*
58. Rider *NUEVO*
59. Snyk *NUEVO*
60. Ambientes de desarrollo para IU *NUEVO*
61. Visual Studio Code
62. VS Live Share *NUEVO*

**EVALUAR**
63. Bitrise *NUEVO*
64. Codefresh *NUEVO*
65. Grafeas *NUEVO*
66. Heptio Ark *NUEVO*
67. Jaeger *NUEVO*
68. kube-bench *NUEVO*
69. Ocelot *NUEVO*
70. Optimal Workshop *NUEVO*
71. Stanford CoreNLP *NUEVO*
72. Terragrunt *NUEVO*
73. TestCafe *NUEVO*
74. Traefik *NUEVO*
75. Wallaby.js *NUEVO*

**RESISTIR**

## LENGUAJES & FRAMEWORKS

**ADOPTAR**

**PROBAR**
76. Jepsen
77. MMKV *NUEVO*
78. MockK *NUEVO*
79. TypeScript

**EVALUAR**
80. Apache Beam *NUEVO*
81. Camunda *NUEVO*
82. Flutter
83. Ktor *NUEVO*
84. Nameko *NUEVO*
85. Polly.js *NUEVO*
86. PredictionIO *NUEVO*
87. Puppeteer *NUEVO*
88. Q# *NUEVO*
89. SAFE stack *NUEVO*
90. Spek *NUEVO*
91. troposphere
92. WebAssembly
93. WebFlux *NUEVO*

**RESISTIR**

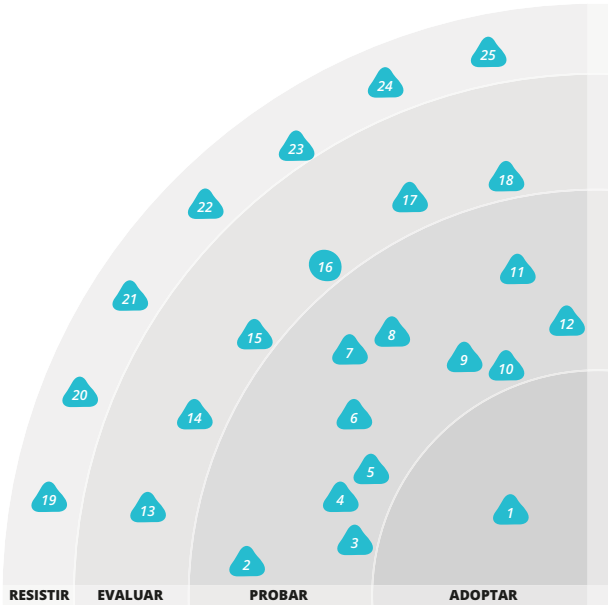
# TÉCNICAS

**ADOPTAR**
1. Event Storming

**PROBAR**
2. 1% canary *NUEVO*
3. Bounded Buy *NUEVO*
4. Crypto shredding *NUEVO*
5. Cuatro métricas clave *NUEVO*
6. Multi-account cloud setup *NUEVO*
7. Observabilidad como código *NUEVO*
8. Estrategia de proporción de riesgo del vendedor *NUEVO*
9. Coste de ejecución como función de aptitud de arquitectura *NUEVO*
10. Secretos como servicio *NUEVO*
11. Security Chaos Engineering
12. Datos de versiones para analíticas reproducibles *NUEVO*

**EVALUAR**
13. Katas de Caos *NUEVO*
14. Distroless Docker images *NUEVO*
15. Entrega incremental con costos *NUEVO*
16. Escaner de Configuración de Infraestructura
17. Verificación de compilación descendente antes del commit *NUEVO*
18. Malla de Servicios

**RESISTIR**
19. Manipulación de clusteres Hadoop con estado usando herramientas de gestión de configuración *NUEVO*
20. Uso genérico de la nube
21. Arquitectura de servicios en capas *NUEVO*
22. Gestión de datos maestros *NUEVO*
23. Envidia de Microservicios
24. Eventos petición-respuesta en flujos de interacción con el usuario *NUEVO*
25. RPA *NUEVO*




Cuando las organizaciones avanzan hacia los microservicios, uno de los motivos principales es la esperanza de un tiempo de comercialización más rápido. Sin embargo, esta aspiración solo tiende a ser alcanzada cuando los servicios (y sus equipos de soporte) están hábilmente repartidos a lo largo de los límites del dominio de negocio de larga duración. De lo contrario, las funcionalidades significativas requerirán, naturalmente, una coordinación estrecha entre múltiples equipos y servicios, introduciendo fricciones naturales al competir por la priorización de la hoja de ruta. La solución a este problema es un buen modelado del dominio, y el **EVENT STORMING** se ha convertido rápidamente en unos de nuestros métodos favoritos para identificar prontamente los conceptos clave en el espacio del problema y alineando una variedad de partes interesadas del mejor modo para encontrar las soluciones posibles.

La retroalimentación rápida es uno de nuestros valores principales para construir software.
Durante muchos años, hemos utilizado la aproximación canary release para fomentar la retroalimentación temprana sobre nuevas versiones de software, y al mismo tiempo reducimos el riesgo desplegando incrementalmente a usuarios seleccionados. Una de las preguntas con respecto a esta técnica es cómo segmentar a los usuarios. Los lanzamientos de versiones canary a un segmento muy pequeño (por ejemplo, 1%) de usuarios pueden ser un catalizador para el cambio. Mientras que comenzar con un segmento muy pequeño de usuarios permite a los equipos sentirse cómodos con la técnica, la captura de comentarios rápidos de los usuarios permite que diversos equipos observen el impacto de los nuevos lanzamientos y puedan aprender y ajustar el rumbo según sea necesario, un cambio que no tiene precio en la cultura de la ingeniería. Llamamos a esto, el poderoso **1% CANARY.**

*Las soluciones out-the-box o SaaS tienden a expandir agresivamente su alcance para enredarse en cada parte de su negocio. Recomendamos una estrategia para solo seleccionar productos de vendedores que son modulares, desacoplados y que puedan ser contenidos dentro del Bounded Context de una sola capacidad comercial.*

(Bounded Buy)

La mayoría de las organizaciones que no tienen los recursos para crear su software a la medida, seleccionarán soluciones out-the-box o SaaS para cumplir con sus requisitos. Sin embargo, con demasiada frecuencia, estas soluciones tienden a expandir agresivamente su alcance para enredarse en cada parte de su negocio. Esto difumina los límites de integración y hace que el cambio sea menos predecible y lento. Para mitigar este riesgo, recomendamos a las organizaciones que desarrollen un modelo de capacidad claro y luego emplear una estrategia que llamamos **BOUNDED BUY** — la cual es solo seleccionar productos de vendedores que son modulares, desacoplados y que contengan Bounded Context una sola capacidad comercial. Esta modularidad e independencia en la entrega deberia ser incluido en el criterio de aceptación en un proceso de selección del proveedor.

Mantener un control apropiado sobre datos sensibles es dificil, especialmente cuando -por propósitos de respaldo y restauración- los datos son copiados fuera del sistema maestro de guardado. **CRYPTO SHREDDING** es la práctica en la que se hace que los datos sensibles sean ilegibles, al sobreescribir o eliminar deliberadamente claves de cifrado utilizadas para protegerlos. Por ejemplo, una tabla completa con los detalles personales de los clientes puede ser encriptada utilizando claves randómicas para cada registro, almacenando estas claves en una tabla diferente. Si un cliente ejerce su "derecho a ser olvidado", simplemente se puede eliminar la clave apropiada para "despedazar (shredding)" efectivamente los datos encriptados. Esta técnica puede ser muy útil cuando se tiene la confianza de mantener un control apropiado sobre un conjunto pequeño de claves de encriptación pero una menor confianza sobre el control de un conjunto de datos más grande.

*El informe State of DevOps y el subsecuente libro Accelerate resaltan un hecho sorprendente: para predecir y mejorar el desempeño de un equipo solo necesitamos medir el tiempo de entrega, frecuencia de implementación, tiempo promedio de restauración (MTTR) y porcentaje de fallos de cambio.*

(Cuatro métricas clave)

El informe State of DevOps , publicado por primera vez en 2014, indica que los equipos de alto rendimiento crean organizaciones de alto rendimiento. Recientemente, el equipo detrás del informe publicó Accelerate , que describe el método científico que utilizaron en el informe. Una de las principales conclusiones de ambas son las **CUATRO MÉTRICAS CLAVE** para respaldar el rendimiento de la entrega del software: tiempo de entrega, frecuencia de implementación, tiempo promedio de restauración (MTTR) y porcentaje de fallos de cambio. Como una consultoría que ha ayudado a muchas organizaciones a transformarse, estas métricas han surgido una y otra vez como una forma de ayudar a las organizaciones a determinar si están mejorando el rendimiento general. Cada métrica crea un ciclo virtuoso y enfoca a los equipos en la mejora continua: para reducir el tiempo de entrega, se reducen las actividades inútiles que, con lo cual, se puenden realizar entregas más frecuentemente; la frecuencia de despliegue obliga a sus equipos a mejorar sus prácticas y automatización; Su velocidad para recuperarse de las fallas se incrementa con las mejores prácticas, la automatización y el monitoreo, lo que reduce la frecuencia de las mismas.

El autoservicio bajo demanda es una característica clave (y un beneficio) de la computación en la nube. Cuando se despliegan entornos de servicios a gran escala utilizando una sola cuenta, las reglas y los procesos relacionados con el uso de esa cuenta se vuelven necesarios y a menudo implican pasos de aprobación que aumentan el tiempo de respuesta. Un mejor enfoque es una configuración de nube con **MÚLTIPLES CUENTAS** donde se utilizan varias cuentas, por ejemplo, una cuenta por equipo. Esto agrega sobrecarga en otros lugares, como, asegurar una facturación compartida, permitir la comunicación entre VPC y administrar la relación con el proveedor de la nube. Sin embargo, a menudo acelera el desarrollo y, por lo general, mejora la seguridad,

ya que las cuentas de con un único servicio son más fáciles de auditar y, en caso de ataque, el impacto se reduce considerablemente. Tener varias cuentas también reduce el acoplamiento, porque una cuenta proporciona un buen límite para los servicios que se pueden mover en bloque a otro proveedor de la nube.

*La observabilidad es una parte integral de la operación de una arquitectura distribuida basada en microservicios. Recomendamos tratar las configuraciones de los sistemas de observabilidad como código.*

(Observabilidad como código)

La observabilidad es una parte integral de la operación de una arquitectura distribuida basada en microservicios. Utilizamos diversas salidas de sistema como trazas distribuidas, logs agregados y métricas para determinar el estado interno de cada uno de los componentes distribuidos y así diagnosticar donde existen problemas y así averiguar su causa raíz. Un aspecto importante en un ecosistema de observabilidad es la monitorización — visualización y análisis de las salidas de los sistemas— y la emisión de alertas cuando se detecten condiciones inesperadas. Tradicionalmente, la configuración de los tableros de monitorización y el establecimiento de alertas se realiza utilizando sistemas basados en interfaces gráficas de usuario. Esta aproximación lleva a que las configuraciones de los tableros no sean repetibles, y la imposibilidad de comTRIAL y ajustar de manera contínua las alertas para evitar fatiga por su recepción o pérdida de alertas importantes, y el alejamiento de las mejores prácticas de las organizaciones. Recomendamos encarecidamente tratar las configuraciones de los sistemas de observabilidad como código, lo que denominamos **OBSERVABILIDAD COMO CÓDIGO,** y adoptar infrastructura como código para la infraestructura de monitorización y alertas. Elija productos de observabilidad que admitan configuración a través de código bajo control de versiones y ejecución de APIs u órdenes con tuberías (pipelines) de entrega contínua (CD). La observabilidad como código es un aspecto que se olvida a menudo en la infraestructura como código y, creemos, que es suficientemente crucial como para ser considerada de manera explícita.

Muchas veces, en un esfuerzo por tercerizar el riesgo a sus proveedores, las empresas buscan "una garganta para estrangular" en sus más cruciales y arriesgadas implementaciones de sistema. Desafortunadamente, esto les ofrece menos elecciónes de soluciones y menos flexibilidad. En lugar de esto, las empresas deberían tratar de mantener la máxima independencia de vendedor donde estén más expuestas a riesgos. Vemos surgir una nueva **ESTRATEGIA DE PROPORCIÓN DE RIESGO DEL VENDEDOR** que alenta la inversión para mantener la independencia del vendedor en los sistemas empresariales más cruciales. Las funciones empresariales menos cruciales pueden aprovecharse de la provisión más dinámica de una solución propia del vendedor ya que les permite absorber con más facilidad el impacto de perder este vendedor. Esta compensación se ha hecho aparente conforme que los proveedores más importantes de la nube han extendido el ámbito de sus ofertas de servicios. Por ejemplo, el uso del AWS Secret Management Service puede agilizar el desarrollo inicial y tiene el beneficio de integración al ecosistema, pero también añadirá más inercia, si alguna vez tengas que cambiar a otro proveedor de nube, de lo que habría añadido si hubieras implementado, por ejemplo, a Vault.

Aún hay equipos que no realizan un seguimiento del coste de ejecución de sus aplicaciones de una forma tan cercana como deberían, a medida que su arquitectura de software o su uso evoluciona. Esto es particularmente cierto en el caso de estar usando una arquitectura serverless, que los desarrolladores suponen que proporcionará menores costes, ya que no se paga por ciclos de servidor no utilizados. Sin embargo, los principales proveedores de servicios en la nube tienen ya suficiente experiencia en la definición de los modelos de precio y las funciones serverless de uso intensivo, si bien son muy útiles para realizar iteraciones rápidamente, pueden encarecerse muy rápido si se las compara con servidores dedicados en la nube (o locales). Recomendamos a los equipos definir el **COSTE DE EJECUCIÓN COMO UNA FUNCIÓN DE APTITUD DE LA ARQUITECTURA** del sistema, lo que significa: realizar un seguimiento del coste de ejecución de los servicios en relación su valor entregado; cuando se observen desviaciones respecto a lo esperado o a lo aceptable, plantear una discusión sobre si ha llegado el momento de evolucionar la arquitectura.

Durante mucho tiempo hemos advertido a las personas sobre la tentación de registrar secretos dentro de sus repositorios de código fuente. Previamente hemos recomendado el desacoplamiento de la gestión de secretos del código fuente. Sin embargo, ahora estamos viendo emerger una serie de buenas herramientas que ofrecen **SECRETOS COMO SERVICIO.** Con este enfoque, en lugar escribir los secretos directamente en el código o configurarlos como parte del ambiente, las aplicaciones los recuperan desde un proceso separado.

Herramientas como Vault de HashiCorp permiten administrar secretos separados de la aplicación y aplicar políticas como el cambio de contraseña frecuente. Aunque en esta edición del Radar hemos tenido, en su mayoría, blips nuevos, creemos que vale la pena seguir mencionando la utilidad de **SECURITY CHAOS ENGINEERING.** Lo hemos movido a Prueba porque los equipos que usan esta técnica confían en que las políticas de seguridad que tienen son lo suficientemente sólidas para manejar los modos comunes de fallos de seguridad. Aún así, proceda con precaución al usar esta técnica, no queremos que nuestros equipos se vuelvan insensibles a estos problemas.

*El análisis de datos a gran escala o de problemas de inteligencia de máquina nos permiten reproducir diferentes versiones de análisis, realizados en diferentes conjuntos de datos y parámetros es inmensamente valiosa.*

(Los datos de versiones para analíticas reproducibles)

Cuando se trata de problemas de análisis de datos a gran escala o de problemas de inteligencia de máquina, la capacidad de reproducir diferentes versiones de análisis, realizados en diferentes conjuntos de datos y parámetros es inmensamente valiosa. Para lograr un análisis reproducible, tanto los datos como el modelo (incluyendo la elección de algoritmos, parámetros e hiperparámetros) deben ser controlados por versiones. **LOS DATOS DE VERSIONES PARA ANALÍTICAS REPRODUCIBLES,** son un problema relativamente más complicado que los modelos de versiones debido al tamaño de los datos. Las herramientas como DVC ayudan en la creación de versiones de datos, al permitir que los usuarios confirmen y envíen archivos de datos a un depósito remoto de almacenamiento en la nube, utilizando un flujo de trabajo similar a git.
Esto facilita que los colaboradores obtengan una versión específica de los datos para reproducir un análisis.

*La repetición de las Katas ayuda a los ingenieros a internalizar sus nuevas habilidades. Combinamos la disciplina de Katas con Técnicas de Ingeniería del Caos para ayudar a los ingenieros a descubrir el problema, recuperar la normalidad tras un error, realizar un análisis postmortem y encontrar la causa.*

(Katas de Caos)

**KATAS DE CAOS** es una técnica que nuestros equipos han desarrollado para entrenar y mejorar las habilidades de los ingenieros de infraestructuras y plataformas. Combina técnicas de Ingeniería del Caos—es decir, provocar errores y cortes de servicio en un entorno controlado—con la enseñanza sistemática y el enfoque en el entrenamiento propio de una Kata. Aquí, por Kata nos referimos a patrones de código que producen errores controlados, permitiendo a los ingenieros descubrir el problema, recuperar la normalidad tras un error, realizar un análisis postmortem y encontrar la causa. La repetición de las Katas ayuda a los ingenieros a internalizar sus nuevas habilidades.

*Al crear imágenes de Docker para nuestras aplicaciones, a menudo nos preocupan dos cosas: la seguridad y el tamaño de la imagen. Con esta técnica, la huella de la imagen se reduce a las dependencias de la aplicación, sus recursos y la ejecución del lenguaje, sin considerar la distribución del sistema operativo.*

(Distroless Docker images)

Al crear imágenes de Docker para nuestras aplicaciones, a menudo nos preocupan dos cosas: la seguridad y el tamaño de la imagen. Tradicionalmente, para la revisión de seguridad en contenedores, hemos utilizado herramientas que detectan y parchan vulnerabilidades y brechas comunes. También distribuciones pequeñas como Alpine Linux para reducir el tamaño de la imágen y mejorar el rendimiento de la distribución. En este Radar, estamos entusiasmados por abordar la seguridad y el tamaño de los contenedores con una nueva técnica llamada **DISTROLESS DOCKER IMAGES,** iniciada por Google. Con esta técnica, la huella de la imagen se reduce a las dependencias de la aplicación, sus recursos y la ejecución del lenguaje, sin considerar la distribución del sistema operativo. Las ventajas de esta técnica incluyen la reducción de ruido en los escaneos de seguridad, menor superficie de ataques de seguridad, la reducción de la sobrecarga de parchar vulnerabilidades y un tamaño de imagen aún menor para un mayor rendimiento. Google ha publicado un conjunto de distroless container images para diferentes lenguajes. Puedes crear imágenes de aplicaciones distroless con la herramienta de compilación de Google, Bazel, que tiene reglas para crear distroless containers o simplemente usa multi-stage Dockerfiles. Hay que tener en cuenta que, por defecto, los distroless containers no tienen una interfaz de comandos para la depuración.

Sin embargo, en internet se puede encontrar fácilmente versiones depurables de distroless containers, incluyendo busybox shell.

En ThoughtWorks, como precursores y líderes en el entorno agile, hemos sido defensores de la práctica de entrega de software incremental. Hemos aconsejado también a muchos clientes para que piensen en el software 'listo para usar' con un enfoque de "¿Puede este software ser puesto en producción incrementalmente?". En muchas ocasiones, esto ha sido muy díficil por el enfoque "big-bang" de la mayoría de los proveedores de software, que usualmente involucra la migración de una grandes cantidades de información. Recientemente, sin embargo, hemos tenido experiencias exitosas utilizando la **ENTREGA INCREMENTAL CON COSTOS,** poniendo a disposición procesos de negocio especificos incrementalmente para un número reducido de usuarios. Nosotros recomendamos que se evalue si se puede aplicar esta práctica con el proveedor de software de su preferencia para ayudar a reducir los riesgos de un enfoque de entrega del tipo big-bang.

*La mayoría de organizaciones establecen de forma predeterminada una configuración cerrada y centralmente administrada para reducir riesgos, pero esto también crea cuellos de botella sustanciales en la productividad. Un enfoque alternativo es permitir a los equipos administrar su configuración, y usar una herramienta para garantizar que la configuración sea realizada de manera acertada y segura.*

(Escaner de configuración de infraestructura)

Por algún tiempo hemos recomendado incrementar el empoderamiento de los equipos sobre la entrega de su stack íntegro, incluyendo la infraestructura. Esto significa incrementar la responsabilidad del equipo en configurar su infraestructura de manera acertada, segura y que cumpla con los estándares. Cuando se ADOPTan estrategias usando la nube, la mayoría de organizaciones establecen de forma predeterminada una configuración cerrada y centralmente administrada para reducir riesgos, pero esto también crea cuellos de botella sustanciales en la productividad. Un enfoque alternativo es permitir a los equipos administrar su configuración, y usar un **ESCANER DE CONFIGURACIÓN DE INFRAESTRUCTURA** para garantizar que la configuración sea realizada de manera acertada y segura. Watchmen es una herramienta interesante, creada para proporcionar seguridad basada en reglas de las configuraciones de una cuenta AWS, que pertenece y es operada independientemente por los equipos de entrega. Scout2 es otro ejemplo de herramienta para el escaneo de configuración para garantizar el cumplimiento de los estándares de seguridad.

En arquitecturas e implementaciones más complejas, puede que no sea inmediatamente obvio que una compilación que depende del código que se está verificando actualmente está rota. Los desarrolladores que intentan arreglar una compilación defectuosa podrían encontrarse trabajando con un objetivo en movimiento, ya que la compilación se desencadena continuamente por las dependencias ascendentes. **LA VERIFICACIÓN DE COMPILACIÓN DESCENDENTE ANTES DEL COMMIT** es una técnica muy sencilla: hacer que un script de pre-commit o pre-push compruebe el estado de estas compilaciones descendentes y alerte al desarrollador de antemano de que una compilación no funciona.

A medida que las organizaciones grandes hacen la transición hacia equipos más autónomos que son dueños de sus microservicios y los operan, ¿cómo pueden garantizar la consistencia y compatibilidad necesarias entre ellos sin depender de una infraestructura centralizada de alojamiento? Para trabajar de forma conjunta y eficiente, incluso los microservicios autónomos se deben alinear con algún estándar organizacional. Una **MALLA DE SERVICIOS** (service mesh) brinda capacidades consistentes de descubrimiento, seguridad, rastreabilidad, monitoreo y manejo de errores, sin necesitar de elementos compartidos como una puerta de enlace de APIs (API gateway) o un bus de servicios. Una implementación típica integra procesos de proxy reversos implementados junto a cada proceso de servicio, posiblemente en contenedores separados. Estos proxy se comunican con registros de servicios, proveedores de identidad, agregadores de registros de eventos y otros servicios. La interoperabilidad y visibilidad de los servicios se obtiene por medio de una implementación compartida del proxy, pero no por compartir una instancia en tiempo de ejecución. Hemos insistido en usar enfoques descentralizados para la gestión de microservicios por algún tiempo y nos agrada ver como emerge este patrón consistente. Los proyectos de código abierto como Linkerd e Istio seguirán madurando y facilitando aún más la implementación de mallas de servicios.

Cuando las organizaciones escogen una distribución de vanilla Hadoop o Spark en lugar de una distribución de proveedor, tienen que decidir cómo quieren provisionar y administrar el cluster. Ocasionalmente, vemos una **MANIPULACIÓN DE LOS CLUSTERS DE HADOOP USANDO HERRAMIENTAS DE GESTIÓN DE CONFIGURACIÓN** como Ansible, Chef y otras. Aunque estas herramientas son excelentes para aprovisionar componentes de infraestructura inmutables, no son muy efectivas cuando deben gestionar sistemas con estado y pueden a menudo requerir un esfuerzo significante intentando manejar y evolucionar los clusters. En su lugar recomendamos utilizar herramientas como Ambari para aprovisionar clusters Hadoop o Spark con estado.

*Cada vez más, vemos que las organizaciones se preparan para usar una "cualquier nube" y evitar el bloqueo de proveedores a toda costa. Su estrategia limita el uso de la nube solo a las características comunes de todos los proveedores de la nube, por lo que se pierden los beneficios exclusivos de cada proveedor.*

(Uso genérico de la nube)

Los principales proveedores de servicios en la nube se han vuelto cada vez más competitivos en sus precios y en el rápido ritmo de lanzamiento de nuevas funcionalidades. Esto deja a los consumidores en un lugar difícil al elegir y comprometerse con un proveedor. Cada vez más, vemos que las organizaciones se preparan para usar una "cualquier nube" y evitar el bloqueo de proveedores a toda costa. Esto, por supuesto, conduce a un **USO GENÉRICO DE LA NUBE.** Vemos organizaciones que limitan su uso de la nube solo a las características comunes de todos los proveedores de la nube, por lo que se pierden los beneficios exclusivos de cada proveedor. Vemos organizaciones que hacen grandes inversiones en capas de abstracción locales que son demasiado complejas de construir y demasiado costosas de mantener para permanecer agnósticos de cada nube. El problema del bloqueo de proveedor es real. Recomendamos abordar este problema con una estrategia de múltiples nubes que evalúe el coste y el esfuerzo de migración de las capacidades de una nube a otra, comparándolos con los beneficios de usar características específicas de una nube concreta. Recomendamos aumentar la portabilidad de las cargas de trabajo enviando las aplicaciones como contenedores Docker ampliamente ADOPTados, utilizando protocolos de identidad y seguridad de código abierto para migrar fácilmente la identidad de las cargas de trabajo, usando una estrategia proporcional al riesgo en cuanto a proveedores para mantener la independencia de la nube solo cuando sea necesario, y usar Polycloud para mezclar y combinar servicios de diferentes proveedores donde tenga sentido. En resumen, cambie su enfoque de un uso genérico de la nube a una estrategia sensata de múltiples nubes.

Una característica que define a una arquitectura de microservicios es que los componentes y los servicios están organizados en función de las capacidades del negocio. Independientemente del tamaño, los microservicios encapsulan una agrupación significativa de funcionalidad e información para permitir la entrega independiente de valor de negocio. Esto contrasta con los enfoques anteriores en arquitectura de servicios que organizaban los servicios de acuerdo con las características técnicas. Hemos observado una serie de organizaciones que han ADOPTado una **ARQUITECTURA DE MICROSERVICIOS EN CAPAS,** que de alguna manera es una contradicción en sí misma. Estas organizaciones han recurrido a la organización de servicios principalmente de acuerdo con un rol técnico, por ejemplo, APIs de experiencia, APIs de proceso o APIs de sistema. Es demasiado fácil asignar a los equipos de tecnología por capas, por lo que la entrega de cualquier cambio de negocio valioso requiere una coordinación lenta y costosa entre varios equipos. Advertimos contra los efectos de esta organización por capas y recomendamos la organización de acorde a los servicios y equipos principalmente de acuerdo con las capacidades de negocio.

**MASTER DATA MANAGEMENT** (MDM) es un clásico ejemplo de solución empresarial "silver bullet": promete resolver una clase de problemas aparentemente relacionados de una sola vez. A través de la creación de un único punto central de cambio, cordinación, pruebas y deployment, las soluciones MDM impactan negativamente la habilidad de una organización de responder a cambios de negocio. Las implementaciones tienden a ser largas y complejas, ya

que las organizaciones tratan de capturar y mapear todo los datos "master" en el MDM mientras integran la solución MDM en los sistemas consumidores o productores.

Los microservicios han surgido como una técnica de arquitectura destacada en los sistemas modernos basados en la nube, pero todavía creemos que los equipos deberían proceder con cuidado al tomar esta decisión. La **ENVIDIA DE MICROSERVICIOS** tienta a los equipos a complicar su arquitectura al tener muchos servicios simplemente por que es una opción arquitectónica de moda. Platformas como Kubernetes facilitan en gran parte el despliegue de conjuntos complejos de microservicios, y los proveedores de servicio están empujando sus soluciones hacia la administración de microservicios, potencialmente llevando a equipos más adentro de este camino. Es importante recordar que los microservicios intercambian complejidad de desarrollo por complejidad operacional y se requiere unas bases sólidas de pruebas automatizadas, entrega continua y cultura de DevOps.

En muchas ocasiones nos hemos encontrado con diseños de sistema que usan e**VENTOS PETICIÓN-RESPUESTA EN FLUJOS DE INTERACCIÓN CON EL USUARIO.** En estos casos, la interfaz de usuario se mantiene bloqueada o el usuario tiene que esperar durante la carga de una nueva página hasta que se recibe el correspondiente mensaje de respuesta a una petición previa. La principal justificación para este tipo de diseños suele ser el rendimiento o una aproximación unificada a la comunicación entre backends en casos de uso síncronos y asíncronos. Creemos que el incremento de complejidad—en desarrollo, testing y operaciones—sobrepasa a los beneficios de tener una aproximación unificada, y sugerimos fuertemente el uso de peticiones HTTP síncronas cuando resulten necesarias. Bien implementada, la comunicación usando HTTP rara vez resulta ser el cuello de botella en un sistema distribuido.

*El problema con esta aproximación centrada solamente en la automatización de los procesos de negocio, sin abordar los sistemas o capacidades de software subyacentes, es que esto puede hacer que sea aún más difícil cambiar los sistemas subyacentes. RPA tiende a introducir un acoplamiento adicional, haciendo que cualquier intento futuro de abordar el entorno de TI heredado sea aún más difícil.*

(RPA)

La automatización de procesos robóticos (**RPA**) es una parte clave de muchas iniciativas de transformación digital, ya que promete obtener ahorros de costes sin tener que modernizar la arquitectura y los sistemas subyacentes.
El problema con esta aproximación centrada solamente en la automatización de los procesos de negocio, sin abordar los sistemas o capacidades de software subyacentes, es que esto puede hacer que sea aún más difícil cambiar los sistemas subyacentes ya que se añade un acoplamiento adicional. Esto hace que cualquier intento futuro de abordar el entorno de TI heredado sea aún más difícil. Muy pocos sistemas pueden permitirse ignorar el cambio y, por lo tanto, los esfuerzos de la RPA deben ir acompañados de estrategias de modernización de los sistemas heredados adecuadas.

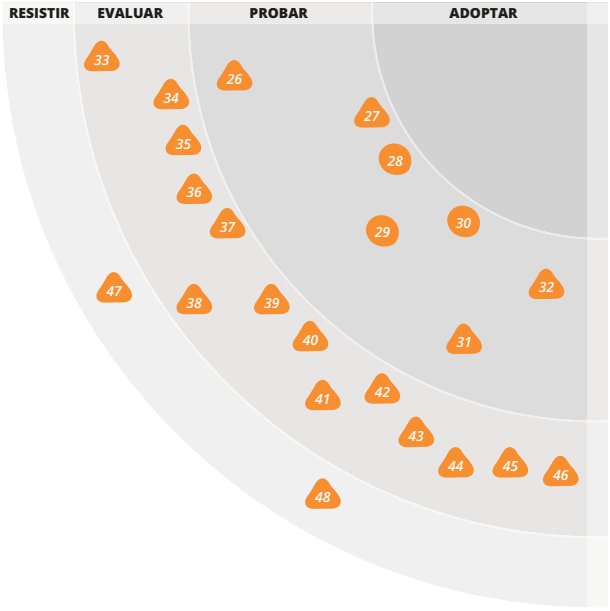
# PLATAFORMAS

**ADOPTAR**

**EVALUAR**
26. Apache Atlas *NUEVO*
27. AWS
28. Azure
29. Contentful
30. Google Cloud Platform
31. VPC Compartida *NUEVO*
32. TICK Stack

**PROBAR**
33. Azure DevOps *NUEVO*
34. CockroachDB *NUEVO*
35. Debezium *NUEVO*
36. Glitch *NUEVO*
37. Google Cloud Dataflow *NUEVO*
38. gVisor *NUEVO*
39. IPFS *NUEVO*
40. Istio *NUEVO*
41. Knative *NUEVO*
42. Pulumi *NUEVO*
43. Quorum *NUEVO*
44. Resin.io *NUEVO*
45. Rook *NUEVO*
46. SPIFFE *NUEVO*

**RESISTIR**
47. Paquetes Data-Hungry *NUEVO*
48. Plataformas de código bajo *NUEVO*




*Las necesidades del manejo de metadatos están creciendo. Atlas provee capacidades para modelar tipos para metadatos, clasificar recursos de datos, hacer seguimiento de jerarquía de datos y permitir el descubrimiento de datos. Esto encaja en las necesidades del manejo de datos de las empresas.*

(Apache Atlas)

Con las crecientes y diversas necesidades de datos de las empresas ha venido un aumento en la necesidad del manejo de metadatos. **APACHE ATLAS** es un framework de manejo de metadatos que encaja en las necesidades del manejo de datos de las empresas. Atlas provee capacidades para modelar tipos para metadatos, clasificar recursos de datos, hacer seguimiento de jerarquía de datos y permitir el descubrimiento de datos. Sin embargo, cuando se construye una plataforma de manejos de metadatos, debemos ser cuidadosos de no repetir los errores de manejo de datos master.

Situamos inicialmente a **AWS** en ADOPTar hace siete años, y la cantidad de servicios y su resiliencia han mejorado a pasos agigantados desde entonces. Sin embargo, movemos AWS a TRIAL, no porque encontremos alguna deficiencia en su oferta, sino porque sus competidores, GCP y Azure han madurado hasta tal punto que, actualmente, escoger un proveedor en la nube se ha vuelto una tarea cada vez más compleja. Reservamos la categoría de ADOPTar para cuando existe una clara plataforma ganadora en la industria. Durante muchos años, AWS ha sido la opción por defecto, pero ahora creemos que las organizaciones deben hacer una selección equilibrada entre los proveedores en la nube, que tenga en cuenta sus ventajas geográficas y regulaciones locales, su alineamiento estratégico con los proveedores (o la falta de este), y por supuesto, que ajuste los productos diferenciadores de estos proveedores con sus necesidades más relevantes.

Microsoft continuamente ha mejorado a **AZURE** y hoy no mucho lo separa de la experiencia núcleo proveída por los proveedores de nube más grandes – Amazon, Google y Microsoft. Los proveedores de la nube parecen estar de acuerdo y buscan diferenciarse entre

sí en otras áreas como funcionalidades, servicios y estructura de costos. Microsoft es el proveedor que ha mostrado real interés en los requisitos legales de empresas europeas. Tienen una estrategia matizada y creíble, incluyendo ofertas únicas como Azure Germany y Azure Stack, que ofrecen un poco de certeza a compañías europeas en anticipación del GDPR y posibles cambios legislativos en los Estados Unidos.

Los sistemas de administración de contenidos implementados solo en el back-end, y accesibles a traves de una API, son conocidos como: "Headless CMS" (CMSes) y se están convirtiendo en un componente común de las plataformas digitales. **CONTENTFUL** es un "headless CMS" moderno que nuestros equipos han podido integrar exitosamente en sus flujos de desarrollo. Nos gusta particularmente su propuesta de "API-first", y la implementación de CMS como código. Ofrece potente soporte para primitivos de modelado para contenidos asi como scripts de evolución para modelos de contenido, lo cual permite que pueda ser tratado como cualquier otro esquema de almacenamiento de datos y poder aplicar las prácticas de diseño evolutivo de bases de datos al desarrollo de CMS. Otras características notables que nos han gustado son la inclusión por defecto de dos CDNs para entregar recursos de multimedia y documentos en JSON, además de buen soporte para localización, y la capacidad - aunque con cierto esfuerzo - de integrarse con Auth0.

Debido a que la plataforma de la nube de Google, **GOOGLE CLOUD PLATFORM** (GCP) se ha expandido en cuanto a disponibilidad geográfica y madurez de sus servicios, los clientes de todo el mundo pueden ahora consideralo seriamente para sus estrategias en la nube. En algunas áreas, GCP ha conseguido igualdad de características con su principal competidor, Amazon Web Services, mientras que en otras áreas se ha diferenciado notablemente con plataformas de machine learning accesibles, herramientas de ingeniería de datos y soluciones viables de servicios como Kubernetes (GKE). En la práctica, nuestros equipos no tienen mas que elogios por la experiencia de desarrollador trabajando con las herramientas y APIs de GCP.

*Un patrón recomendado es usar una red privada virtual en la nube gestionada a nivel organizacional y dividida en subredes de menor tamaño bajo el control de cada equipo de entrega. Una VPC compartida convierte organizaciones, proyectos, VPCs y subredes en entidades de primer nivel en las configuraciones de red –esto simplifica la configuración y hace más transparente la seguridad y el control de acceso.*

(VPC Compartida)

Así como hemos ganado más experiencia con la nube pública entre grandes y pequeñas organizaciones, ciertos patrones han surgido. Uno de esos patrones es una red privada virtual en la nube gestionada a nivel organizacional y dividida en subredes de menor tamaño bajo el control de cada equipo de entrega. Esto está estrechamente relacionado con la idea de configuración multicuenta para la nube y ayuda a particionar una infraestructura entre equipos. Después de explícitamente instalar esta configuración varias veces usando VPCs, subredes, grupos de seguridad y NACLs, nos gusta mucho la noción de Google de **VPC COMPARTIDA**. Una VPC compartida convierte organizaciones, proyectos, VPCs y subredes en entidades de primer nivel en las configuraciones de red. Las VPCs pueden ser gestionadas por los administradores de una organización, los cuales pueden delegar la administración de subredes a proyectos. De esta manera los proyectos pueden ser explícitamente asociados con subredes en la VPC. Esto simplifica la configuración y hace más transparente la seguridad y el control de acceso.

**TICK STACK** es una colección de componentes de código abierto que se combinan entregando una plataforma para fácilmente almacenar, visualizar y monitorizar datos de series temporales como métricas y eventos. Los componentes son: Telegraf, un agente servidor para recolectar y reportar métricas; InfluxDB, una base de datos de alto rendimiento para series temporales; Chronograf, una interfaz de usuario para la

plataforma; y Kapacitor, un motor de procesamiento de datos que puede procesar, transmitir y agrupar datos desde InfluxDB. A diferencia de Prometheus, que está basado en un modelo pull, TICK Stack está basado en un modelo push para recolectar datos. El corazón del sistema es el componente InfluxDB, que es una de las mejores bases de datos para series temporales. El stack está respaldado por InfluxData y aunque se necesite la versión empresarial para funcionalidades tales como el clustering de bases de datos, sigue siendo una elección bastante buena para la monitorización. Nosotros la estamos usando en algunos lugares en producción y hemos obtenido buenas experiencias.

**AZURE DEVOPS** incluye un conjunto de servicios tales como repositorios Git alojados, pipelines para CI y CD y repositorios de artefactos. Azure DevOps ha reemplazado a Visual Studio Team Services. Hemos tenido una buena experiencia iniciando proyectos rápidamente con Azure DevOps: manejando, construyendo y desplegando aplicaciones para Azure. También hemos tenido algunos retos—como la falta de un soporte completo a pipelines como código de CI y CD, tiempos lentos de compilación e inicio del agente, separación de la compilación y publicación en pipelines diferentes—y hemos experimentados algunos tiempos de inactividad. Estamos esperando que los servicios de Azure DevOps mejoren con el tiempo para que brinden una buena experiencia a los desarrolladores cuando hacen hosting de aplicaciones en Azure, y que brinden una experiencia sin dificultades al integrarse con otros servicios de Azure.

*Inspirado en el documento técnico de Google sobre Spanner, su base de datos distribuida basada en relojes atómicos, CockroachDB es una alternativa de código abierto que proporciona transacciones distribuidas y geo-particiones sin dejar de admitir SQL.*

(CockroachDB)

**COCKROACHDB** Es una base de datos distribuida de código libre inspirada en la publicación oficial Spanner: La base de datos distribuida de Google . En CockroachDB, los datos se dividen automáticamente en rangos, usualmente 64MB, y son distribuidos a través de nodos en el clúster. Cada rango tiene un grupo de consenso y porque utilizan el algoritmo de consenso de Raft, los datos siempre se mantienen sincronizados. Con su diseño único, CockroachDB proporciona transacciones distribuidas y geo-particionadas mientras aún soporta SQL. A diferencia de Spanner, que se basa en TrueTime con reloj atómico para la linealización, CockroachDB usa NTP para la sincronización del reloj y proporciona serialización como el nivel de aislamiento por defecto. Si estás trabajando con datos estructurados que encajan en un solo nodo, entonces elije una base de datos relacional tradicional. Sin embargo, si tus datos necesitan escalar a través de nodos, ser consistentes y sobrevivir fallas, entonces te recomendamos mirar más de cerca a CockroachDB.

*Siempre estamos a la búsqueda de herramientas o plataformas que puedan transmitir cambios de base de datos. Debezium es una excelente elección. Funciona reaccionando a los cambios en los logs de las bases de datos y es altamente escalable y resiliente a fallas.*

(Debezium)

**DEBEZIUM** es una plataforma de captura de datos modificados (o change data capture, CDC) que puede transmitir cambios de base de datos a topics de Kafka. La CDC es una técnica popular con múltiples casos de uso, incluyendo la replicación de datos a otras bases de datos, alimentando a sistemas de analítica, extrayendo microservicios de monolitos e invalidando cachés. Siempre estamos a la búsqueda de herramientas o plataformas en este espacio (Hablamos acerca de Bottled Water en un Radar previo) y Debezium es una excelente elección. Usa un enfoque de CDC basado en logs, lo que significa que funciona reaccionando a los cambios en los logs de las bases de datos. Debezium usa Kafka Connect, lo que lo hace altamente escalable y resiliente a fallas, y tiene conectores CDC para múltiples bases de datos, incluyendo Postgres, Mysql y MongoDB. Lo estamos usando en algunos de nuestros proyectos y ha funcionado muy bien para nosotros.

Nos ha intrigado **GLITCH**, un entorno de desarrollo en línea colaborativo que te permite copiar y adaptar (o "remezclar") fácilmente las aplicaciones web

existentes o crear las tuyas propias. Arraigado en el espíritu del "manitas", es ideal para personas que están aprendendiendo a programar, aunque tiene la capacidad de soportar aplicaciones más complejas. El foco principal está en JavaScript y Node.js, pero tiene soporte limitado para otros lenguajes. Con integración de edición en directo, alojamiento, uso compartido y versionado de código automático, Glitch ofrece una visión refrescante y diferente de la programación colaborativa.

**GOOGLE CLOUD DATAFLOW** es útil en escenarios de ETL tradicionales para leer datos de una fuente, transformarlos y almacenarlos en un recipiente, con configuraciones y escalamiento administrados por el flujo de datos. Dataflow soporta a Java, Python y Scala y provee *wrappers* para conexiones a various tipos de fuentes de datos. Sin embargo, la versión actual no te permitirá librerias adicionales, lo cual puede hacerlo inapropiado para ciertas manipulaciones de datos. Tampoco puedes cambiar el DAG (grafo aciclico dirigido o directed acyclic graph) de flujo de datos. Debido a eso, si tu ETL tiene flujos de ejecución condicional basados en parametros, tal vez no podrás usar a dataflows sin soluciones alternas.

**GVISOR** es un kernel de espacio de usuario (user-space) para contenedores. Éste limita la superficie accesible a la aplicación del kernel anfitrión sin delimitar todas las funcionalidades que se esperan. A diferencia de otras tecnologías de sandbox, tal como el hardware virtualizado (KVM y Xen) o la ejecución basada en reglas como (seccomp, SELinux y AppArmor), gVisor toma un enfoque distinto para el sandboxing de contenedores, interceptando las llamadas al sistema de la aplicación y actuando como el sistema operativo invitado sin la necesidad de la traducción de instrucciones mediante hardware virtualizado. gVisor incluye una Iniciativa de Open Container (OCI) de tiempo de ejecución llamada runsc que se integra con Docker y provee soporte experimental a Kubernetes. gVisor es un proyecto relativamente nuevo y recomendamos evaluarlo para su panorama de seguridad de contenedores.

En muchos casos, blockchain no es el lugar correcto para almacenar un archivo blob (ejemplo: imagen o audio). Cuando se esta desarrollando DApp, una opción es poner los archivos blob en alguna forma centralizada de almacenamiento fuera del blockchain, lo cual usualmente demuestra falta de confianza. Otra opción es almacenarlos en InterPlanetary File System (**IPFS**), el cual es un sistema de archivos orientado al contenido, versionado y punto a punto. Está diseñado para distribuir altos volúmenes de datos con alta eficiencia y desconectado de cualquier dominio centralizado. Los archivos están almacenados en pares que no necesitan confiar entre sí. IPFS mantiene cada versión de un archivo de manera que nunca se pierdan archivos importantes. Hemos visto IPFS como un buen complemento de la tecnología blockchain. Más allá de su aplicación en blockchain, IPFS tiene el objetivo ambicioso de descentralizar la infraestructura de internet.

Al crear y operar un ecosistema de microservicios, una de las preguntas iniciales a responder es como implementar temas transversales como el descubrimiento de servicios, la seguridad de servicio a servicio y de origen a servicio, la capacidad de observación (incluida telemetría y rastreo distribuido), lanzamientos continuos y elasticidad. En el último par de años, nuestra respuesta por defecto a esta pregunta ha sido usando la técnica malla de servicios. Una malla de servicios ofrece la implementación de estas capacidades transversales en forma de capa de infraestructura que se configura como código. Las políticas de configuración pueden ser aplicadas consistentemente a todo el ecosistema de microservicios; aplicado tanto dentro como fuera del tráfico de malla (vía el proxy de malla como puerta de enlace) así como en el tráfico en cada servicio (vía el mismo proxy de malla como un contenedor adicional). Mientras nos mantenemos mirando de cerca el progreso de diferentes proyectos de código libre de malla de servicios, como Linkerd, hemos utilizado con éxito **ISTIO** en producción con un modelo operativo sorprendentemente fácil de configurar.

Como desarrolladores de aplicaciones, nos encanta concentrarnos en resolver problemas del negocio y

dejar que la plataforma subyacente se ocupe de las aburridas pero difíciles tareas de desplegar, escalar y gestionar aplicaciones. A pesar de que la arquitectura serverless es un paso en esa dirección, la mayoría de ofertas populares están ligadas a una implementación propietaria, lo cual significa un proveedor fijo. **KNATIVE** intenta abordar esto siendo una plataforma serverless de código abierto que se integra bien con el popular ecosistema de Kubernetes. Con Knative es posible modelar el procesamiento bajo demanda en uno de los frameworks soportados (incluyendo Ruby on Rails, Django y Spring entre otros); suscribirse, suministrar y gestionar eventos; integrarse con herramientas de CI y CD familiares; y construir contenedores a partir del código fuente. Proporciona un conjunto de componentes middleware para construir aplicaciones centradas en el código y basadas en contenedores, que pueden ser elásticamente escalables. Knative es una plataforma atractiva que merece ser considerada para sus necesidades serverless.

*Si bien se centra en las arquitecturas nativas de la nube, Pulumi es una herramienta de automatización de infraestructura que se distingue por permitir que las configuraciones se escriban en TypeScript / JavaScript, Python y Go.*

(Pulumi)

Estamos interesados en **PULUMI**, un herramienta prometedora para la automatización de infraestructuras nativas en la nube. Pulumi se diferencia por admitir que las configuraciones seas escritas en TypeScript/JavaScript, Python y Go — no requiere YAML. Pulumi está enfocado en arquitecturas nativas en la nube, incluidos contenedores, funciones sin servidor y servicios de datos, además de proporcionar un buen soporte para Kubernetes.

Ethereum es el ecosistema líder en tecnología blockchain. Hemos visto la aparición de soluciones que tienen por objetivo expandir esta tecnología en entornos empresariales que normalmente requieren de permisos de acceso a la red y privacidad en las transacciones, así como un mejor rendimiento y menor latencia. **QUORUM** es una de estas soluciones. Desarrollada originalmente por J.P. Morgan, Quorum se autoposiciona como "una versión empresarial de Ethereum". A diferencia de un nodo Hyperledger Burrow, que ha creado una nueva maquina virtual Ethereum (EVM), el código de Quorum es una derivación del código oficial del cliente de Ethereum, de modo que puede evolucionar de manera conjunta con Ethereum. Aunque conserva muchas de las caracteristicas de la base de datos (ledger) de Ethereum, Quorum cambia el protocolo de consenso PoW (proof of work) original, por otros más eficientes y añade soporte para transacciones privadas. Con Quorum, los desarrolladores pueden emplear sus conocimientos de Ethereum, como por ejemplo el lenguaje Solidity y los contratos Truffle para construir aplicaciones blockchain empresariales. Sin embargo, basándonos en nuestra experiencia, Quorum no está aún listo para su adopción empresarial; por ejempo, carece de control de acceso para contratos privados, no funciona bien con balanceadores de carga, y solo tiene un soporte de base de datos parcial, lo que conduce a un esfuerzo significativo de despliegue y diseño. Recomendamos ser prudentes a la hora de implementar Quorum, mientras nos mantenemos atentos a la evolución de su desarrollo.

**RESIN.IO** es una herramienta de internet of Things (IoT) que solo hace una cosa y la hace bien: despliega contenedores en los dispositivos. Los desarrolladores usan un portal SaaS para gestionar los dispositivos y asignar aplicaciones definidas por Dockerfiles. La plataforma puede construir contenedores para varios tipos de hardware y desplegar imágenes al vuelo. Para contenedores, Resin.io usa balena, un engine basado en el framework Moby creado por Docker. La plataforma permanece en desarrollo, tiene algunas aristas por trabajar y le faltan algunas funcionalidades (ejemplo, trabajo con registros privados), pero la actual lista de características incluye la opción de hacer ssh a un contenedor en un dispositivo desde un portal web, tornándose en un futuro prometedor.

**ROOK** es un orquestrador de almacenamiento nativo en la nube de código abierto para Kubernetes. Rook se integra con Ceph y brinda sistemas de almacenamiento de archivos, bloques y objetos dentro del cluster de Kubernetes, ejecutándolos sin interrupciones al mismo tiempo que otras aplicaciones y servicios que esten consumiento el almacenamiento. Usando operadores de Kubernetes, Rook orquesta a Ceph en

el plano de control y se aleja del camino de datos entre aplicaciones y Ceph. El almacenamiento es uno de los componentes más importantes de la computación nativa a la nube y creemos que Rook, aunque aun es un proyecto de incubación de la CNCF, nos acerca un poco más a la auto-suficiencia y la portabilidad en la nube pública y en los despliegues locales.

Permitir que elementos clave de la plataforma innovadora y de alta escala de Google estén disponibles como ofertas open-source parece haberse convertido en una tendencia. De la misma forma en que HBASE fue inspirada en BigTable y Kubernetes fue inspirado en Borg, **SPIFFE** esta aprovechando LOAS de Google para concebir un concepto crítico para "cloud-native" denomidado identidad de carga de trabajo (workload identity). Los estándares de SPIFFE están respaldados por OSS SPIFFE Runtime Environment (SPIRE), el mismo que automáticamente entrega identidades criptográficamente verificables para cargas de trabajo de software. Mientras SPIRE no esta totalmente listo para su uso en ambientes de producción, nosotros vemos un gran valor de manera independiente de la plataforma, para hacer poderosas verificaciones de identidad entre cargas de trabajo en infraestructuras de TI modernas y distribuidas. SPIRE soporta muchos casos de uso, incluyendo traducción de identidad, autenticación de cliente OAuth, mTLS "encryption everywhere" y observación de carga de trabajo. Istio utiliza SPIFFE por defecto.

**DATA-HUNGRY PACKAGES** son soluciones que requieren la absorción de datos en sí mismos para funcionar. En algunos casos, incluso pueden requerir convertirse en "master" de esos datos. Una vez que los datos son propiedad del paquete, ese software se convierte en la única forma de actualizar, cambiar o acceder a ellos. Los paquetes data-hungry pueden resolver un problema particular de negocio, como un ERP. Sin embargo, las demandas de datos de inventario o finanzas en una organización a menudo requieren una integración compleja y cambios en los sistemas que se encuentran fuera del alcance original.

Las **PLATAFORMAS DE BAJO CÓDIGO** utilizan interfaces de usuario y configuraciones gráficas para crear aplicaciones. Por desgracia, los entornos de bajo código suelen promoverse con la idea de que eliminan la necesidad de tener equipos de desarrollo cualificados. Estas propuestas ignoran el hecho de que escribir código es solo una pequeña parte de todo lo que debe suceder para crear programas de calidad -- prácticas como el control de versiones, el testeado y un diseño cuidadoso de las soluciones son igual de importantes. Si bien estas plataformas son adecuadas para ciertos casos de uso, sugerimos tratarlas con precaución, especialmente cuando vienen acompañadas de promesas extravagantes como reducción de coste y aumento de la productividad.

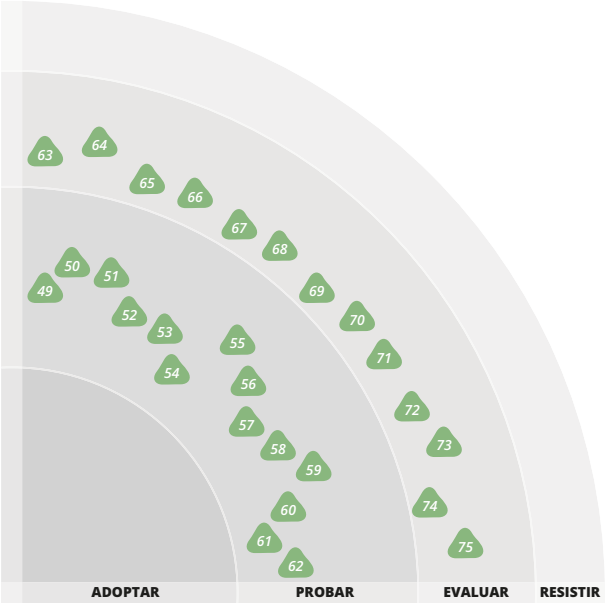
# HERRAMIENTAS

**ADOPTAR**

**PROBAR**
49. acs-engine *NUEVO*
50. Archery *NUEVO*
51. ArchUnit
52. Cypress
53. git-secrets *NUEVO*
54. Headless Firefox
55. LocalStack *NUEVO*
56. Mermaid *NUEVO*
57. Prettier *NUEVO*
58. Rider *NUEVO*
59. Snyk *NUEVO*
60. Ambientes de desarrollo para IU *NUEVO*
61. Visual Studio Code
62. VS Live Share *NUEVO*

**EVALUAR**
63. Bitrise *NUEVO*
64. Codefresh *NUEVO*
65. Grafeas *NUEVO*
66. Heptio Ark *NUEVO*
67. Jaeger *NUEVO*
68. kube-bench *NUEVO*
69. Ocelot *NUEVO*
70. Optimal Workshop *NUEVO*
71. Stanford CoreNLP *NUEVO*
72. Terragrunt *NUEVO*
73. TestCafe *NUEVO*
74. Traefik *NUEVO*
75. Wallaby.js *NUEVO*

**RESISTIR**




Azure Container Service Engine (**ACS-ENGINE**) es un generador de plantillas para el administrador de recursos de Azure (Azure Resource Manager, ARM). Las configuraciones necesarias del cluster estan definidas en un archivo JSON; acs-engine lee estas cluster definitions y genera varios archivos que pueden ser consumidos por ARM. Esta herramienta también provee la flexibilidad para escoger diferentes orquestadores — incluyendo Kubernetes, DC/OS, OpenShift, Swarm mode y Swarm y configurar sus características y agentes del cluster. Hemos estado utilizando acs-engine en varios proyectos y lo recomendamos para administrar clusters en Azure Container Service.

Estamos viendo avances significativos en la integración de herramientas de seguridad con los procesos modernos de entrega de software. **ARCHERY** es una herramienta de código libre con una comunidad activa que está haciendo un buen trabajo integrando una colección de otras herramientas, incluyendo a Zap. Diseñado principalmente para aplicaciones web, Archery hace que sea fácil integrar herramientas de seguridad dentro de sistemas de construcción y despliegue. Sus tableros también permiten hacer tanto seguimiento de vulnerabilidades como escaneo de red y aplicaciones.

**ARCHUNIT** es una librería Java para testing, que permite revisar características de arquitectura, tales como paquetes y dependencias de clases, verificación de anotaciones e incluso layer consistency. Nos gusta que se ejecuta como pruebas unitarias dentro de la configuración existente para pruebas, sin embargo solo soporta arquitecturas basadas en Java. El ArchUnit test suite puede ser incorporado en un ambiente CI, o ser deployado por un pipeline, haciendo esto más fácil de implementar fitness functions en un evolutionary architecture way.

Ejecutar pruebas “end-to-end” puede ser retador, debido a la larga duración del processo de ejecución, la fragilidad de algunas pruebas y los retos de corregir fallas en CI cuando las pruebas se están ejecutando en modo “headless”. Nuestros equipos han tenido muy buenas experiencias con **CYPRESS** resolviendo

algunos problemas comunes como el bajo desempeño y largos tiempos de respuesta en llamada y carga de recursos. Cypress es una herramienta útil que ayuda a los desarrolladores a construir pruebas "end-to-end" y grabar todos los pasos de la pruebas como un video MP4 para hacer más fácil el proceso de identificación de errores.

*Una herramienta simple que le impide ingresar contraseñas y otra información confidencial a un repositorio git, también escanea todas las revisiones históricas antes de hacer público un repositorio.*

(git-secrets)

La seguridad sigue siendo de importancia capital, y la introducción accidental de credenciales y otros secretos en los repositorios de control de versiones es un importante vector de ataque. **GIT-SECRETS** es una herramienta simple que impide introducir contraseñas y otra información confidencial en un repositorio git. También puede escanear todas las revisiones históricas antes de hacer público un repositorio, si queremos asegurarnos de que nunca se haya introducido una credencial accidentalmente. git-secrets viene con soporte incorporado para las claves y credenciales comunes de AWS y también puede configurarse rápidamente para otros proveedores.

**HEADLESS FIREFOX** tiene la misma madurez que Headless Chrome para las pruebas de front-end. Al igual que con Headless Chrome, con Firefox en modo "headless" podemos disfrutar de las pruebas del navegador sin los componentes visibles de la interfáz de usuario (UI), ejecutando el conjunto de pruebas de UI mucho más rápido.

*Probar los servicios en la nube localmente es un desafío. LocalStack resuelve este problema para AWS al proporcionar implementaciones dobles de prueba local de una amplia gama de servicios de AWS, incluidos S3, Kinesis, DynamoDB y Lambda.*

(LocalStack)

Uno de los retos de usar los servicios de la nube es poder desarrollar y TRIAL localmente usando estos servicios. **LOCALSTACK** resuelve este problema en AWS al proveer implementaciones de dobles de test locales para un amplio rango de servicios AWS, incluyendo S3, Kinesis, Dynamodb y Lambda. Se basa en algunas de las mejores herramientas como Kinesalite, Dynalite y Moto y añade procesos aislados y funcionalidades de inyección de errores. LocalStack es muy fácil de usar y se entrega con un simple runner de JUnit y una extensión para JUnit 5. Lo estamos usando en algunos de nuestros proyectos y nos ha impresionado.

**MERMAID** te deja generar diagramas desde un lenguaje similar a markdown. Nacido desde la necesidad de simplificar documentación, Mermaid ha crecido hasta convertirse en un gran ecosistema con plugins para Confluence, Visual Studio Code y Jekyll nombrando unos pocos. Para ver cómo funciona, puedes usar el Live Editor en GitHub. Mermaid también tiene una conveniente interfaz de comando que permite generar ficheros SVG, PNG y PDF como resultado de definition files. Hemos usado Mermaid en muchos proyectos y nos gusta la simplicidad de describir gráficos y diagramas de flujo con lenguaje markdown y revisar los archivos de definición con el código de repositorio.

*Un formateador de código automatizado para JavaScript, Prettier aumenta la coherencia y la legibilidad y reduce el esfuerzo del desarrollador tanto en formatear como en participar en debates de equipo inútiles sobre el estilo del código.*

(Prettier)

**PRETTIER** es un formateador de código dogmático y automatizado de JavaScript (creciendo soporte en otros lenguajes). Al obligar un estilo de formateo, incrementa la consistencia, legibilidad y reduce esfuerzo de desarrollo para formatear o tener debates desgargantes con el equipo acerca de estilo de código. Es probable que discrepes con el estilo que Prettier obliga; sin embargo, los beneficios que le darán a tu equipo son superiores a problemas de estilo que puedas tener con esta herramienta. Prettier puede ser usado como un precommit hook o como plugin en tu ambiente de desarrollo. Como cualquier formateador, un reformateo único de código fuente confundiría a tu controlador de versiones pero es una desventaja menor. Particularmente, nos gusta como Prettier cambia la estrategia basada en linters pues, tomando la iniciativa de gofmt, asegura que el mismo tenga un estilo válido siempre, en lugar de sólo validarlo.

Hemos hablado de Visual Studio Code en el Radar desde 2015, pero no es el único ambiente de desarrollo multi plataforma .NET en la industria. Recientemente, **RIDER**, parte de la plataforma IDEA desarrollada por JetBrains, ha ganado adopción, especialmente con aquellas personas acostumbradas a la velocidad y destreza brindadas por ReSharper, que maneja la refactorización en Rider. Sin embargo, Rider hace más que ReSharper para brindar la plataforma completa de IDEA e incrementar la productividad de desarrollo. Sin importar cual sea tu plataforma preferida, vale la pena explorar Rider pues, por el momento, le lleva ventaja a Visual Studio Code. Además, es bueno ver este ecosistema vivo, a la vez que la competencia entre estas herramientas aseguran su mejora.

**SNYK** ayuda a encontrar, reparar y monitorear vulnerabilidades conocidas en los árboles de dependencias de npm, Ruby, Python, Scala, Golang, .NET, PHP, Java y Docker. Cuando se agrega al pipeline de construcción, Snyk supervisa y prueba continuamente el árbol de dependencias de librerías, contra una base de datos de vulnerabilidades y sugiere la actualización de la versión de la dependencia, mínima necesaria, para su corrección.

Mientras más y más equipos acogen el DesignOps, las prácticas y herramientas en este espacio también maduran. Muchos de nuestros equipos ahora trabajan con lo que podría llamarse **AMBIENTES DE DESARROLLO PARA IU,** que proporcionan un ambiente integral para iterar rápidamente sobre componentes de IU, enfocándose en la colaboración entre los diseñadores de experiencia de usuario y los desarrolladores. Actualmente contamos con algunas opciones en este espacio: Storybook, react-styleguidist, Compositor y MDX. Estas herramientas pueden utilizarse de manera aislada a la hora de desarrollar una librerìa de componentes o el diseño de un sistema, o embebidas dentro de un proyecto de aplicación web. En vez de tener que levantar toda la aplicación, más un BFF, más servicios para simplemente añadir una caracteristica a un componente, podemos levantar el servidor de desarrollo de Storybook.

**VISUAL STUDIO CODE** es el entorno de desarrollo gratuito de Microsoft, que además es multiplataforma. Hemos tenido una buena experiencia usándolo para el desarrollo de interfaces de usuario con React y TypeScript, así como con lenguajes de servidor como GoLang, sin necesidad de cambiar entre editores diferentes. Las herramientas, lenguajes soportados y las extensiones mejoran en forma continua. En particular, nos gustaría TRIAL VS Live Share para colaboración y programación remota en pareja en tiempo real. Mientras proyectos complejos en lenguajes tipados estáticos, como Java, .NET o C++ probablemente encontrarán mejor soporte en entornos de desarrollo más maduros de Microsoft o JetBrains, creemos que Visual Studio Code se está transformando cada vez más en una opción para equipos de desarrollo de infraestructura e interfaces de usuario.

*La colaboración en tiempo real con VS Live Share facilita el emparejamiento remoto. En particular, nos gusta que les permita a los desarrolladores colaborar con su propio editor preconfigurado.*

(VS Live Share)

**VS LIVE SHARE** is a suite of extensions for Visual Studio Code and Visual Studio. The real-time collaboration for editing and debugging of code, voice calls, sharing a terminal and exposing local ports have reduced some of the obstacles we'd otherwise encounter when pairing remotely. In particular, we like that Live Share allows developers to collaborate with each other, while continuing to use their preconfigured editor, which includes themes, key maps and extensions.

*La creación, prueba y despliegue de aplicaciones móviles implica una serie de pasos complejos, especialmente para una canalización desde los repositorios de código fuente a las tiendas de aplicaciones. Esta herramienta específica de dominio fácil de configurar puede reducir la complejidad y la sobrecarga de mantenimiento.*

(Bitrise)

Construir, TRIAL y desplegar aplicaciones móviles implica varios pasos complejos, especialmente cuando consideramos pipelines que cubren desde el repositorio de código hasta las tiendas de aplicaciones. Todos estos pasos pueden ser automatizados con scripts y se pueden construir pipelines con herramientas genéricas de CI/CD (Integración Continua/Entrega Continua).

Sin embargo, para equipos que se enfocan en desarrollo móvil, y no requieren integrar (o muy poco) con pipelines de sistemas back-end, una herramienta específica de este dominio puede reducir la complejidad y costos de operación. **BITRISE** es fácil de configurar y provee un conjunto completo de pasos predefinidos para la mayoría de necesidades del desarrollo móvil.

**CODEFRESH** es un servidor de CI alojado similar a CircleCI o a Buildkite. Está basado en contenedores, convirtiendo a los archivos Dockerfile y a los *clusters* de alojamiento de contenedores en entidades de primera clase. Nos gusta que la herramienta promueve un enfoque basado en la entrega a través de *pipelines* y soporta la creación y fusión de ramas (*branches*). Los primeros reportes de nuestros equipos son positivos, pero aún debemos ver cómo funciona para proyectos más grandes y *pipelines* más complejos.

*Continuamente buscamos herramientas y técnicas que permitan a los equipos de entrega trabajar independientemente del resto de una organización más grande mientras se mantienen dentro de sus barandas de seguridad y riesgo. Grafeas es una herramienta de este tipo.*

(Grafeas)

Estamos constantemente en búsqueda de herramientas y técnicas que permitan a los equipos de entrega trabajar independientemente del resto de una larga organización mientras se mantienen seguros y libre de riesgos. **GRAFEAS** es una de estas herramientas. Este permite a las organizaciones publicar metadatos autoritarios de artefactos de software—imágenes de Docker, librerías, paquetes—pudiendo ser luego accedidos desde scripts de construcción u otros controles automatizados. Los mecanismos de control permiten la separación de responsabilidad entre los equipos que publican aprobaciones o vulnerabilidades y los equipos que construyen y despliegan software. A pesar de que algunas organizaciones, como Google y JFrog, usan Grafeas en sus flujos de trabajo, hay que notar que esta herramienta aún se encuentra en versión alpha.

*Como una herramienta para administrar la recuperación de desastres para los clusters y volúmenes persistentes de Kubernetes, Ark es fácil de usar y configurar, y le permite hacer copias de seguridad y restaurarlos a través de una serie de puntos de control.*

(Heptio Ark)

**HEPTIO ARK** es una herramienta para la gestion de recuperacion de desastres de clusters Kubernetes y volúmenes persistentes. Ark es fácil de usar y configurar, y permite respaldar y restaurar clusters a través de una serie de puntos de control. Con Ark se puede reducir significativemente el tiempo de recuperación en caso de un fallo de infraestructura, migrar fácilmente recursos de Kubernetes de un cluster a otro y replicar un ambiente de producción para pruebas y solución de problemas. Soporta los principales proveedores de almacenamiento (incluyendo AWS, Azure y Google Cloud) y, desde la versión 0.6.0, un sistema de extensiones que añade compatibilidad para plataformas de respaldo y volúmenes de almacenamiento adicionales. Los entornos de Kubernetes gestionado, como GKE, proporcionan estos servicios por defecto. De cualquier modo, tanto si operas Kubernetes en tu propio centro de datos como si lo haces en la nube, deberías valorar el uso de Heptio Ark para la recuperación de desastres.

**JAEGER** es un sistema de trazabilidad distribuida de código abierto. Similar a Zipkin, fue inspirado por la publicación de Google Dapper y cumple con OpenTracing. Jaeger es proyecto de código abierto más joven que Zipkin, pero ha ganado popularidad rápidamente debido a que soporta un gran número de lenguajes para las librerías cliente y su fácil instalación en Kubernetes. Hemos usado Jaeger exitosamente con Istio, integrando trazas de aplicación con Envoy en Kubernetes, y utilizando su interfaz de usuario. Y ahora que Jaeger se une a CNCF, anticipamos una mayor participación e integración con otros proyectos de CNFC.

**KUBE-BENCH** es un ejemplo de un escáner de configuración de infraestructura que comprueba de manera automatizada la configuración de Kubernetes con la referencia CIS para K8s. Entre otras áreas, incluye la autenticación de usuarios, los permisos y manejo de datos seguros. Nuestros equipos han encontrado que kube-bench aporta mucho valor para identificar vulnerabilidades en las configuraciones.

**OCELOT** es una puerta de enlace de APIs desarrollada en .NET. Luego de tres años de desarrollo, Ocelot ha conseguido tener un conjunto de características relativamente completo y una comunidad activa. A pesar de que hay muchas excelentes puertas de enlace para APIs (por ejemplo, Kong), la comunidad de .NET parece preferir Ocelot al construir microservicios. Esto es en parte porque Ocelot se integra bien con el ecosistema de .NET (por ejemplo, con IdentityServer). Otra razón puede ser que la comunidad de .NET ha extendido Ocelot para soportar protocolos de comunicación como gRPC, Orleans y WebSocket.

La investigación de experiencia de usuario (User Experience UX) demanda de recolección y análisis de datos para tomar mejores decisiones acerca de los productos que se necesita desarrollar. **OPTIMAL WORKSHOP** es una suite de herramientas que ayuda a realizar esto digitalmente. Características, como primer-click o clasificación de tarjetas, ayudan a validar prototipos y mejorar la navegación de un sitio web, al igual que la presentación de información. Particularmente, para equipos distribuidos el beneficio de Optimal Workshop es que les permite realizar investigación remota.

*La extracción de información empresarial significativa de datos de texto es una técnica clave para el procesamiento de datos no estructurados. Stanford CoreNLP, un conjunto de herramientas de procesamiento de lenguaje natural basado en Java, nos ayuda a utilizar las últimas investigaciones en el campo de la PNL para resolver diversos problemas comerciales*

(Stanford CoreNLP)

Cada vez tenemos más proyectos que requieren procesamiento de datos no estructurados. Poder extraer información significativa de negocio a partir de datos en texto es una técnica clave. **STANFORD CORENLP** es un conjunto de herramientas para el procesamiento de lenguaje natural basadas en Java. Soporta el reconocimiento de named-entity, extracción de relaciones, análisis de sentimientos, clasificación de texto y multiples lenguajes incluyendo inglés, chino y árabe. También las encontrámos útiles para etiquetar corpus y modelos de entrenamiento en nuestros escenarios. Con Stanford CoreNLP, pudimos utilizar las últimas investigaciones en el campo del procesamiento del lenguaje natural (PLN) para resolver diversos problemas de negocio.

Usamos Terraform extensamente como código para configurar una infraestructura de nube. **TERRAGRUNT** es un wrapper delgado para Terraform que implementa las prácticas abogadas por el libro: Terraform: Up and Running. Hemos visto que Terragrunt es muy útil ya que incentiva el uso de modulos versionados y la reusabilidad para diferentes ambientes con características prácticas, incluyendo la ejecución recursiva de código en subdirectorios. Nos gustaría ver evolucionar esta herramienta para soportar prácticas de CD de forma nativa, donde todo el código puede ser empaquetado, versionado y reutilizado a través diferentes entornos en pipelines de CD. Hoy, nuestros equipos logran esto mediante soluciones alternas.

Nuestros equipos nos informan de que han tenido bastante éxito con **TESTCAFE**, que es una herramienta para automatizar pruebas en navegadores, basada en JavaScript. TestCafe permite escribir pruebas en JavaScript o TypeScript y ejecutarlas en cualquier navegador que soporte JavaScript. TestCafe incluye de serie varias funcionalidades útiles como la ejecución paralelizada ya configurada y la simulación de solicitudes HTTP. TestCafe usa un modelo de ejecución asíncrona sin tiempos de espera explícitos, lo que tiene como resultado conjuntos de casos de prueba mucho más estables.

**TRAEFIK** es un proxy reverso de código abierto *(open-source)* y un balanceador de carga. Si estás buscando un proxy que proporcione enrutamiento sencillo sin todas las características de NGINX y HAProxy, Traefik es una muy buena elección. El enrutador provee configuración sin recarga, métricas, monitoreo y cortacircuitos que son esenciales cuando se ejecutan microservicios. También se integra muy bien con Let's Encrypt para proveer terminación SSL. A diferencia de Traefik, las herramientas como NGINX y HAProxy pueden requerir de complementos adicionales para poder aplicar plantillas de configuración en respuesta a escalamiento, adición o remoción de microservicios y puede que necesiten, eventualmente, ser reiniciados, lo que puede ser molesto en ambientes de producción.

Todos estamos obsesionados con obtener *feedback* rápido durante el desarrollo guiado por pruebas (Test Driven Development) y siempre estamos buscando nuevas formas de acelerar dicho *feedback*. **WALLABY.JS** es una extensión comercial disponible para los editores más usados que provee ejecución continua de las pruebas unitarias para JavaScript, resaltando los resultados en cada línea junto a tu código. La herramienta identifica y ejecuta el conjunto mínimo de pruebas relacionadas con el cambio realizado en el código mientras tecleas.

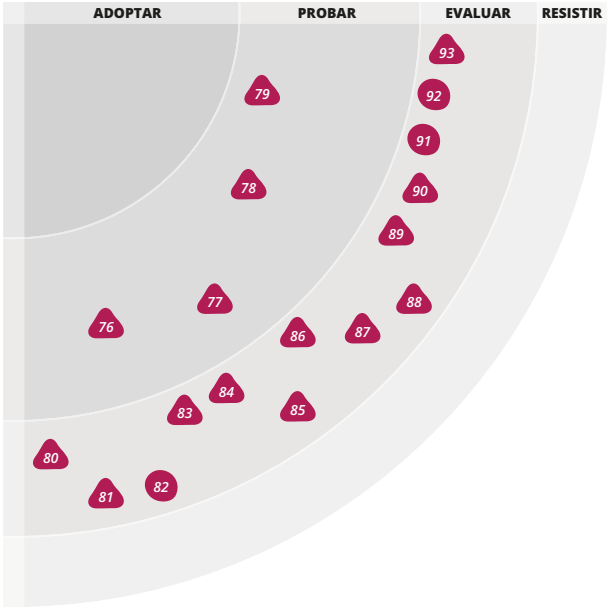
# LENGUAJES & FRAMEWORKS

**ADOPTAR**

**PROBAR**
76. Jepsen
77. MMKV ***NUEVO***
78. MockK ***NUEVO***
79. TypeScript

**EVALUAR**
80. Apache Beam ***NUEVO***
81. Camunda ***NUEVO***
82. Flutter
83. Ktor ***NUEVO***
84. Nameko ***NUEVO***
85. Polly.js ***NUEVO***
86. PredictionIO ***NUEVO***
87. Puppeteer ***NUEVO***
88. Q# ***NUEVO***
89. SAFE stack ***NUEVO***
90. Spek ***NUEVO***
91. troposphere
92. WebAssembly
93. WebFlux ***NUEVO***

**RESISTIR**




Con la creciente adopción de la arquitectura basada en microservicios, estamos construyendo más aplicaciones distribuidas que antes. Aunque son muchos los beneficios de tener una arquitectura desacoplada, la complejidad y el esfuerzo necesarios para comTRIAL la correctitud general del sistema se ha incrementado de manera dramática. **JEPSEN** ofrece herramientas muy necesarias para verificar la correctitud en la coordinación de planificadores de tareas, TRIAL la consistencia eventual, linealizabilidad y serializabilidad características de las bases de datos distribuidas. Hemos usado Jepsen en algunos proyectos y nos gusta el hecho de que permite TRIAL las configuraciones, inyectar y corregir fallos y verificar el estado del sistema tras su recuperación.

Un framework de código libre desarrollado por WeChat, **MMKV** proporciona un rápido almacenamiento clave-valor para aplicaciones móviles. Utiliza las funciones de mapeo de memoria de iOS para evitar la necesidad de guardar explícitamente los cambios y es extremadamente rápida y eficiente. En el caso de un bloqueo inesperado, MMKV permite que la aplicación restaure los datos rápidamente.

*Como biblioteca nativa, MockK ayuda a nuestros equipos a escribir pruebas claras y concisas para las aplicaciones de Kotlin en lugar de usar envoltorios incómodos de Mockito o PowerMock.*

(MockK)

**MOCKK** es una librería para escribir mocks en Kotlin. Su principal filosofía es proveer soporte de primera clase para funcionalidades en el lenguaje Kotlin como Coroutines o bloques lambda. Como librería nativa, ayuda a nuestros equipos a escribir código limpio y conciso para las pruebas en aplicaciones de Kotlin en lugar de utilizar wrappers incómodos de Mockito o PowerMock.

**TYPESCRIPT** es un lenguaje cuidadosamente pensado que nos contínua impresionando con la mejora de sus herramientas y el soporte de los IDE. Con un buen repositorio de definiciones de tipos de TypeScript, podemos aprovechar todas las librerías JavaScript existentes con el beneficio añadido del tipado seguro. Algo particularmente importante a medida que crece

nuestra base de código para el navegador. El tipado seguro de TypeScript permite que el IDE y otras herramientas tengan un mayor contexto del código, y que sea posible realizar cambios y refactorizar de forma fiable. El que TypeScript sea un superconjunto de JavaScript, su documentación y comunidad han ayudado a suavizar la curva de aprendizaje.

**APACHE BEAM** es un modelo de prgramación unificado de código libre para definir y ejecutar tareas de procesamiento de datos en grupo o en tiempo real. Beam proporciona una API portátil para describir estas pipelines independientemente de los motores de ejecución como Apache Spark, Apache Flink o Google Cloud Dataflow. Los diferentes motores de ejecución tienen diferentes capacidades y ofrecer una API portátil es una tarea difícil. Beam intenta encontrar un equilibrio delicado extrayendo las innovaciones de estos motores de ejecución a sus modelos y además trabaja con la comunidad para influenciar en la hoja de ruta de estos motores de ejecución. Beam tiene un conjunto completo de transformaciones de I/O incorporadas que cubren la mayoría de las necesidades de los flujos de datos y también proporciona mecanismos para implementar transformaciones personalizadas para casos de uso específicos. La API portátil y la extensibilidad de las transformaciones IO constituyen un caso convincente para evaluar Apache Beam para las necesidades de las pipelines de datos.

*Aunque tendemos a ser escépticos acerca de las herramientas de modelo de proceso de negocios y de notación (BPMN), Camunda es fácil de probar, versión y refactor. La integración con Spring y Spring Boot lo convierte en una opción sólida.*

(Camunda)

Tendemos a ser bastante escépticos con respecto al modelo de procesos de negocios y las herramientas de notación (BPMN) en general, ya que a menudo se las asocia con entornos de low-code y sus desventajas. Aunque el framework OSS BPMN **CAMUNDA** proporciona algo de ostentosidad, también ofrece flujos de trabajo y motores de decisión que pueden integrarse directamente como una biblioteca en su código Java. Esto facilita las pruebas, versionamiento y los flujos de trabajo de refactor. Camunda también se integra con Spring y Spring Boot, entre otros frameworks, por lo que es una opción sólida.

**FLUTTER** es un framework multiplataforma que permite escribir aplicaciones móviles nativas en Dart. Se beneficia de Dart y puede compilarse en código nativo y comunicarse con la plataforma destino sin puente y cambio de contexto - algo que puede causar cuellos de botella de rendimiento en frameworks como React Native o Weex. La función de recarga en caliente de Flutter es impresionante y proporciona una retroalimentación visual super rápida cuando se edita el código. Actualmente Flutter todavía está en versión beta, pero seguiremos vigilándolo para ver cómo evoluciona su ecosistema.

Kotlin ya no es solo una excelente opción para el desarrollo de aplicaciones móviles. Han surgido nuevas herramientas y frameworks que demuestran tambien el valor de este lenguaje para el desarrollo de aplicaciones web. **KTOR** es uno de esos frameworks. A diferencia de otros frameworks web que soportan Kotlin, Ktor está escrito en Kotlin, utilizando características de lenguaje como corrutinas, lo que permite la implementación asíncrona sin bloqueos. La flexibilidad para incorporar diferentes herramientas de log, inyección de dependencias o un motor de plantillas, además de su arquitectura liviana, hace de Ktor una opción interesante para nuestros equipos para crear servicios RESTful.

Uno de los descubrimientos que hemos hecho después de hablar con nuestros equipos es que Python está volviendo en muchos dominios de tecnología. De hecho, está en pleno proceso de convertirse en el lenguaje más usado. En parte, esto es debido su adopción por parte de científicos de datos y en el ámbito de Machine Learning, pero se ha visto también en equipos que están en proceso de construir microservicios. **NAMEKO** es un framework ultraligero de microservicios y una alternativa a Flask para escribir servicios. A diferencia de Flask, el soporte de Nameko está limitado a Websocket, HTTP y AMQP. Tambiém nos ha gustado su enfoque en testabilidad. Si no se necesitan muchas funcionalidades como las plantillas que ofrece Flask, vale la pena echar un vistazo a Nameko.

**POLLY.JS** es una herramienta simple que ayuda a los equipos a realizar pruebas de sitios y aplicaciones JavaScript. A nuestros equipos de desarrolladores les gusta especialmente que les permite interceptar

y simular interacciones HTTP, lo que hace mas fácil y rápidas las pruebas de código JavaScript sin tener que activar servicios o componentes dependientes.

**PREDICTIONIO** es un servidor de código abierto para machine-learning. Los desarrolladores y científicos de datos pueden usarlo para construir aplicaciones inteligentes de predicción. Como todas las aplicaciones inteligentes, PredictionIO tiene tres partes: recolección de datos y almacenamiento, entrenamiento de modelos, despliegue de modelos y exposición de servicios. Los desarrolladores pueden enfocarse en implementar lógica de procesamiento de datos, algoritmos modelo y lógica de predicción basados en las interfaces apropiadas y liberarse del almacenamiento de datos y entrenamiento de modelos. En nuestra experiencia, PredictionIO puede soportar pequeños y grandes volúmenes de datos de baja concurrencia. Principalmente utilizamos PredictionIO para crear un servicio predictivo para pequeñas y medianas empresas o como prueba de concepto al crear motores de predicción más complejos y personalizados.

En el Radar anterior mencionamos Headless Chrome para los test de front-end. Con la adopción de Chrome DevHERRAMIENTAS Protocol (CDP) por parte de otros navegadores, está surgiendo un nuevo conjunto de librerías para automatización y pruebas en el navegador. CDP permite un control preciso sobre el navegador incluso en modo headless. Están apareciendo nuevas bibliotecas de alto nivel que usan CDP para automatización y pruebas. **PUPPETEER** es una de estas nuevas librerías. Puede conducir un headless-Chrome mediente una aplicación single-page, obtener trazas de tiempo para diagnósticos de rendimiento y más. Nuestros equipos opinan que es más rápido y también más flexible que las alternativas basadas en WebDriver.

*Mientras esperamos que llegue el hardware, podemos experimentar con la computación cuántica utilizando lenguajes y simuladores. Las muestras de Q# pueden darle una idea del futuro potencial de la programación.*

(Q#)

La computación cuántica actualmente se encuentra en una "dimensión desconocida" ya que esta disponible para pruebas sin estar completamente lista. Mientras seguimos esperando que llegue el hardware, podemos experimentar y aprender de lenguajes y simuladores. Aunque IBM y otros han tenido un buen progreso, hemos prestado atención especial a los esfuerzos de Microsoft basados en el lenguaje **Q#** y su simulador (32 qubits localmente y 40 en Azure). Si deseas comenzar a jugar con el posible futuro de la programación, echa un vistazo a su conjunto de muestras en GitHub.

El **STACK SAFE**—acrónimo de Suave, Azure, Fable, y Elmish—reúne un conjunto de tecnologías en un stack coherente para el desarrollo web. Está construido usando el lenguaje de programación F#, tanto en el lado del servidor como en el navegador, enfocándose por lo tanto en la programación funcional y de tipado de datos seguros con un enfoque asíncrono. Ofrece características de productividad como recarga en caliente y permite sustituir partes del stack, por ejemplo, el framework del lado del servidor o el proveedor de la nube.

La adopción de un nuevo lenguaje típcamente genera la necesidad urgente de nuevas herramientas que soporten prácticas de ingeniería maduras como la automatización de pruebas. Kotlin no es una excepción. **SPEK** es un framework de pruebas inspirado en herramientas conocidas como Cucumber, RSpec y Jasmine que escriben pruebas en Gherkin y Specification, permitiendo a los equipos introducir prácticas maduras como behaviour-driven development en el espacio de Kotlin.

Estamos probando **TROPOSPHERE** como herramienta para definir infraestructura como código en AWS para nuestros proyectos en los cuales AWS CloudFormation es utilizado en lugar de Terraform . troposphere es una librería de Python que permite definir programáticamente las descripciones de CloudFormation en formato JSON.
Lo atractivo de troposphere es que facilita la captura temprana de errores de JSON, aplicar verificación de tipos, pruebas unitarias y composición DRY de recursos de AWS.

**WEBASSEMBLY** es un gran paso adelante en las capacidades del navegador como entorno de ejecución de código. Compatible con todos los principales navegadores y sus versiones anteriores, es un formato de compilación binaria diseñado para ejecutarse en el navegador a velocidad de nativos. Abre las opciones de lenguajes que se pueden usar para hacer la funcionalidad del front-end con enfoque temprano en C, C++ y Rust, y también es un objetivo de compilación de LLVM. Cuando se ejecuta en un sandbox, puede interactuar con Javascript y comparte los mismos permisos y modelo de seguridad. Cuando se usa con el nuevo compilador para streaming de Firefox, el resultado es una inicialización más rápida de la página. Aunque aún es muy pronto, el estándar W3C es definitivamente uno de los estándares para empezar a explorar.

*Después de trabajar con un estilo funcional-reactivo en una serie de aplicaciones, nuestros equipos quedaron impresionados e informaron que el enfoque mejora la legibilidad del código y el rendimiento del sistema.*

(WebFlux)

Spring Framework 5 fue lanzado hace más de un año, ADOPTando reactive streams, un estándar para procesamiento asíncrono de transmisión con contrapresión sin bloqueo. El módulo **WEBFLUX** introduce una alternativa reactiva al modulo tradicional Spring MVC para escribir aplicaciones web en el ecosistema de Spring. Después de trabajar con él en varias aplicaciones, nuestros equipos han quedado impresionados y han reportado que el enfoque reactivo (funcional) mejora la lectura del código y el rendimiento del sistema. Ellos notan, sin embargo, que ADOPTar WebFlux requiere un cambio significativo en el pensamiento y recomiendan tener esto en cuenta en la decisión a elegir WebFlux sobre Spring MVC.

Sé el primero en saber cuando el Technology radar sea lanzado, y mantente al tanto de webinars y contenido exclusivo.

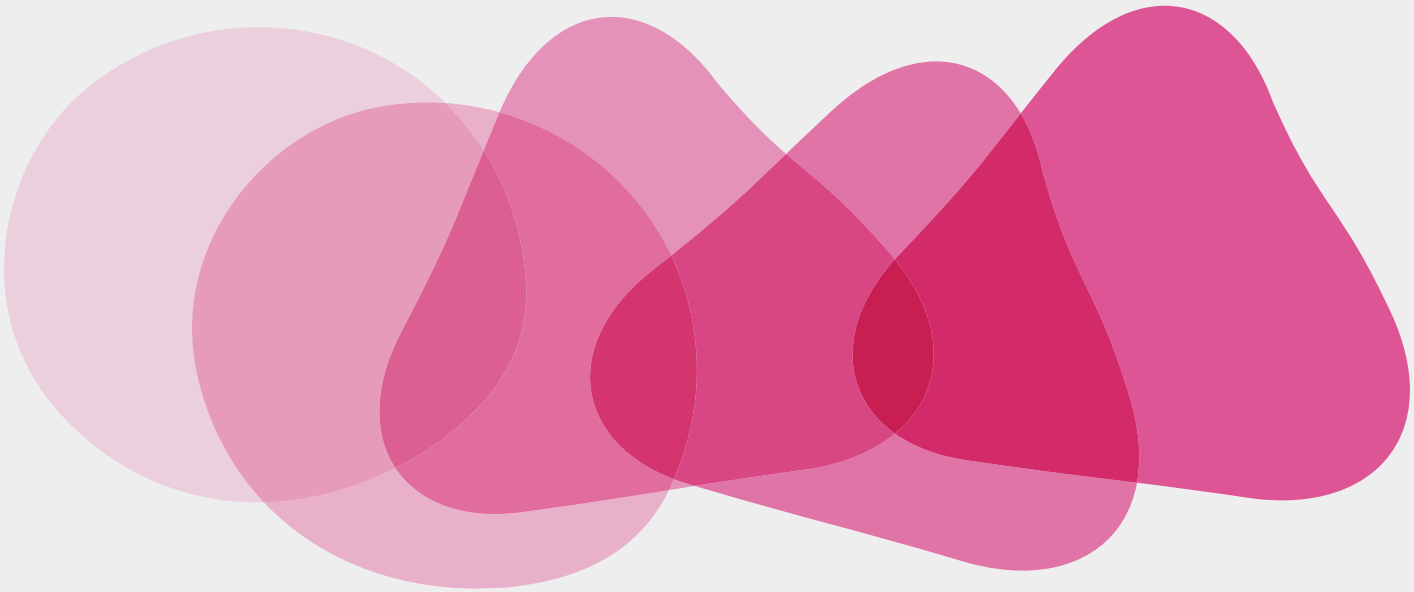
***SUSCRÍBETE AHORA***

***thght.works/Sub-ES***

ThoughtWorks®

ThoughtWorks es una consultora de tecnología y una comunidad de individuos apasionados guiados por propósitos. Ayudamos a nuestros clientes a incorporar la tecnología en el corazón de su negocio, y juntos creamos software relevante para ellos. Dedicados al cambio social positivo; nuestra misión es mejorar a la humanidad a través del software, y nos asociamos con varias organizaciones que luchan en la misma dirección.

Fundada hace 25 años, ThoughtWorks se ha convertido en una compañía de más de 5000 personas, incluyendo una división de productos que crea herramientas pioneras de software para equipos. ThoughtWorks cuenta con 41 oficinas en 14 países: Australia, Brasil, Canada, Chile, China, Ecuador, Alemania, India, Italia, Singapur, España, Tailandia, el Reino Unido y los Estados Unidos.



*thoughtworks.com/es/radar*

***#TWTechRadar***