



DevOps



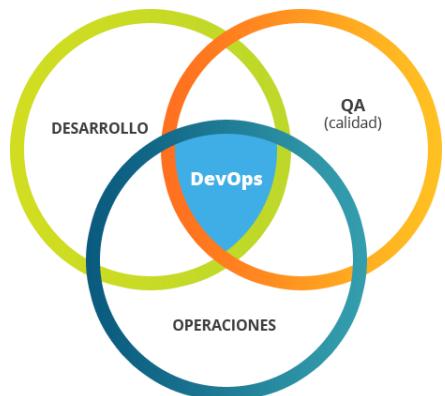
© JMA 2020. All rights reserved

INTRODUCCIÓN

© JMA 2020. All rights reserved

Introducción

- El término DevOps como tal se popularizó en 2009, a partir de los “DevOps Days” celebrados primero en Gante (Bélgica) y está fuertemente ligado desde su origen a las metodologías ágiles de desarrollo software.
- DevOps está destinado a denotar una estrecha colaboración entre lo que antes eran funciones puramente de desarrollo, funciones puramente operativas y funciones puramente de control de calidad.
- Debido a que el software debe lanzarse a un ritmo cada vez mayor, el antiguo ciclo de desarrollo-prueba-lanzamiento de “cascada” se considera roto. Los desarrolladores también deben asumir la responsabilidad de la calidad de los entornos de prueba y lanzamiento.



© JMA 2020. All rights reserved

Introducción

- DevOps establece una “intersección” entre Desarrollo, Operaciones y Calidad, pero no se rige por un marco estándar de prácticas, es una cultura o movimiento que permite una interpretación mucho más flexible en la medida en que cada organización quiera llevarlo a la práctica, según su estructura y circunstancias.
- El objetivo final de DevOps es minimizar el riesgo de los cambios que se producen en las entregas y dar así un mayor valor tanto a los clientes como al propio negocio.
- La característica clave de DevOps es derribar las tradicionales barreras entre los desarrolladores, testers y especialistas en operaciones (Sistemas / Infraestructura) para que trabajen en colaboración a través de herramientas compartidas, corrigiendo diferencias entre personas y entre objetivos, creando vínculos más cercanos entre los participantes con objetivos en común. Además, incorporar el feedback de los clientes/usuarios en el proceso de desarrollo para acelerar la respuesta a errores y mejoras.

© JMA 2020. All rights reserved

Principios

- Según DASA™ (DevOps Agile Skills Association) DevOps presenta 6 principios necesarios para la entrega de servicios TIC.
- Estos principios son:
 1. Acción centrada en el cliente (valor de actuación, innovación)
 2. Crear con el fin en mente (pensamiento de producto y servicio, mentalidad de ingeniero, colaboración)
 3. Responsabilidad Extremo a Extremo (gestión de gastos, nicho, soporte de rendimiento)
 4. Equipos autónomos interfuncionales (perfiles TI, habilidades complementarias)
 5. Mejora continua (fallos rápidos, experimentos)
 6. Automatizar todo lo que se pueda (mejora de la calidad, maximizar el flujo)

© JMA 2020. All rights reserved

Principios

- Acción centrada en el cliente, que se manifiesta por medio del coraje para exponer las disfunciones y liderar cambios; y en la cultura abierta de innovación necesaria para realizar los cambios. Es imprescindible maximizar el retorno de la inversión existente en software, a la vez que innovar y adoptar nuevas tecnologías.
- Crear teniendo en cuenta el objetivo final, lo que requiere una visión amplia de producto y servicio, y no sólo de proyecto, una colaboración entre todas las partes implicadas desde la concepción hasta la operación.
- Responsabilidad extremo a extremo (end to end), hace que todos los participantes se sientan igualmente responsables del producto en todas sus etapas, en lugar de poner foco en completar su parte y olvidarse de los demás. A efectos prácticos esto se traduce en trabajo en equipo, combinar el desarrollo y las operaciones para crear un equipo unilateral que se enfoque en la entrega de objetivos comunes.

© JMA 2020. All rights reserved

Principios

- Equipos autónomos multidisciplinares (cross functional), no es posible tener la visión y responsabilidad extremos a extremo si no se cuenta con equipos capacitados para hacerse cargo del ciclo de vida completo del producto. Además, estos equipos necesitan tener tanto la competencia como la autonomía necesaria para hacerse realmente responsables de su trabajo.
- Mejora continua. El entorno abierto de innovación mencionado anteriormente requiere un contexto, y sobre todo una actitud que fomente la mejora continua, y el inconformismo. Mejorar requiere experimentar, no tener miedo al fracaso sino aprender de él y medir, que es la única forma de evaluar si se produce o no la mejora buscada.
- Automatizar todo lo que se pueda. La aproximación que promueve DevOps, requiere dedicar los recursos disponibles a las actividades que ofrezcan más valor, dejando de lado las que no lo aporten o automatizando todo lo posible. De esa manera, las personas se centran en lo verdaderamente importante y valioso para el objetivo de la organización.

© JMA 2020. All rights reserved

Retos culturales

- Para poder realizar el cambio cultural que supone la implementación del modelo DevOps la organización tiene que ser capaz de trabajar en los siguientes factores:
 - Aprendizaje continuo:
 - Los equipos se auto-organizan, facilitan sesiones de aprendizaje, ...
 - Experimentación:
 - Ofrecer tiempo, entornos aislados (sandbox), eliminar barreras, fallo controlado
 - Calidad en cada versión del producto o servicio:
 - Responsabilidad del producto entregado, automatización
 - Cultura de ingeniero:
 - Ambiciones compartidas por el equipo, experimentación
 - Cultura de efectividad:
 - El fin en mente, retrospectivas, filosofía Lean
 - Cultura de pensamiento centrado en el producto o servicio:
 - Feedback de usuario, story boards
 - Cultura de toma de responsabilidades:
 - No explicar ‘cómo’, explicar ‘qué’, transparencia, pensamiento centrado en el cliente

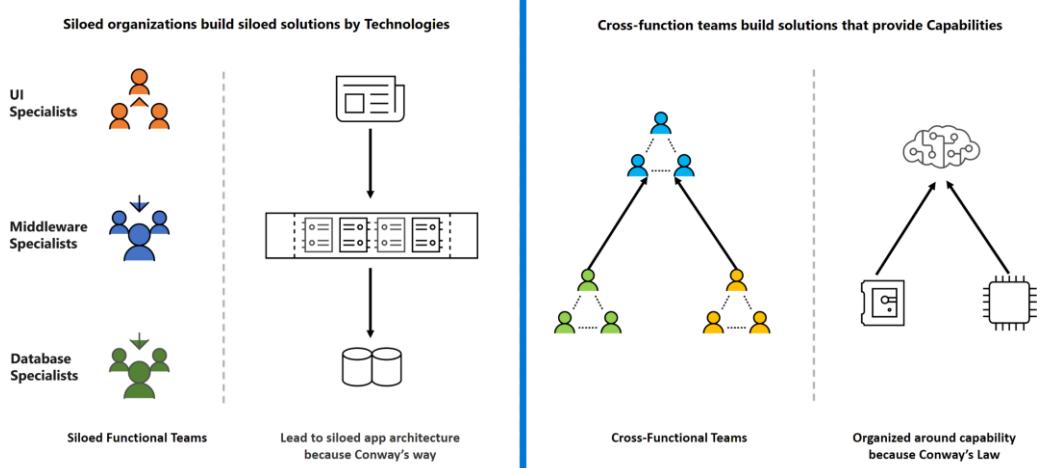
© JMA 2020. All rights reserved

Topologías de equipos

- La distribución de roles, responsabilidades y confianza entre los equipos de TI central, los equipos de las aplicaciones y el equipo de QA es primordial para la transformación operativa involucrada al adoptar DevOps.
- Los equipos de TI se esfuerzan por mantener el control. Los propietarios de aplicaciones buscan maximizar la agilidad. QA busca maximizar la calidad. El equilibrio que se establece en última instancia los diferentes objetivos influye en gran medida en el éxito del modelo DevOps.
- Ley de Conway: cualquier organización que diseña un sistema genera un diseño cuya estructura es una copia de la estructura de comunicación de la organización.
- Según la ley de Conway, los equipos generan arquitecturas basadas en su estructura de comunicación. Comprender este principio es fundamental a medida que trabaja para lograr el equilibrio necesario entre autonomía, control y calidad.

© JMA 2020. All rights reserved

Topologías de equipos



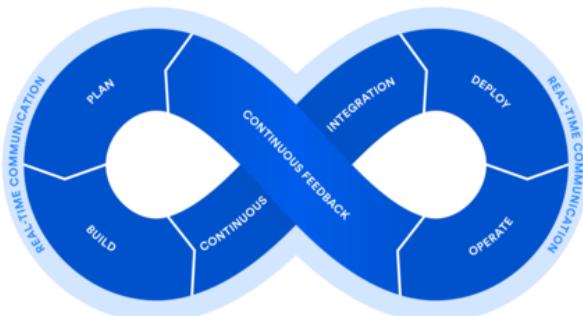
© JMA 2020. All rights reserved

Topologías de equipos

- La solución de Conway es burlar la ley de Conway. Si la organización sigue una estructura determinada para producir servicios y productos y está buscando optimizar, debe replantear la estructura organizativa: evolucionar el equipo y la estructura organizativa para lograr la arquitectura deseada.
- Este principio conduce a topologías de equipo diseñadas intencionalmente en las que los equipos son responsables del extremo a extremo de las aplicaciones, sistemas o plataformas que poseen para lograr la disciplina completa de DevOps.
- Desde una perspectiva de DevOps, las organizaciones deben optimizar la respuesta rápida a las necesidades del cliente. Los equipos que poseen, diseñan e implementan sus aplicaciones y sistemas encuentran su mayor nivel de autonomía en las arquitecturas con las siguientes características:
 - Arquitectura evolucionista que admite cambios constantes
 - Implementabilidad
 - Capacidad de prueba

© JMA 2020. All rights reserved

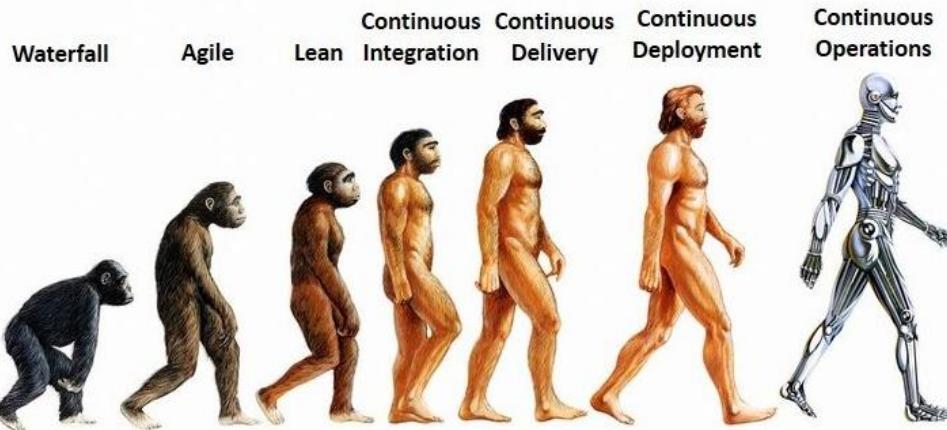
Ciclo de vida



- Planificar: identificar qué funcionalidad se quiere resolver
- Construir: el desarrollo puro, escribir código, pruebas unitarias y documentar lo que se vea necesario
- Integración Continua: automatizar desde el código hasta el entorno de producción
- Desplegar: automatizar el paso a producción
- Operar: vigilar el correcto funcionamiento del entorno, monitorizar
- Feed back: verificar que la funcionalidad tiene valor para el usuario o el retorno esperado

© JMA 2020. All rights reserved

Movimiento DevOps



© JMA 2020. All rights reserved

Ciclo “continuo”

- Muchas empresas se encuentran en algún lugar entre la cascada y las metodologías ágiles. DevOps realmente comienza donde está el “Lean” en la imagen. A medida que se eliminan los cuellos de botella y se comienza a brindar coherencia, podemos comenzar a avanzar hacia la integración continua, la entrega continua y, tal vez, hasta la implementación y las operaciones continuas.
- Con la integración continua, los desarrolladores necesitan escribir pruebas unitarias y se crea un proceso de construcción automatizado. Cada vez que un desarrollador verifica el código, se ejecutan automáticamente las pruebas unitarias y, si alguna de ellas falla, la construcción completa falla. Esta es una mejora con respecto al modelo anterior porque se introducen menos errores en el control de calidad y la compilación solo contiene código de trabajo que reduce la acumulación de defectos.
- Al automatizar las pruebas unitarias en los procesos de construcción, eliminamos muchos errores humanos, mejorando así la velocidad y confiabilidad.

© JMA 2020. All rights reserved

Ciclo “continuo”

- La entrega continua CD (continuous delivery) hace referencia a entregar las actualizaciones a los usuarios según estén disponibles sobre una base sólida y constante.
- La despliegue continuo CD (continuous deployment) es la automatización del despliegue de la entrega continua, que no exista intervención humana a la hora de realizar el despliegue en producción.
- Llegar a CI y CD son el objetivo que muchas empresas se esfuerzan por cumplir y están aprovechando el DevOps para ayudarles a lograrlo. Sin embargo, algunas empresas deben ir más allá porque pueden realizar entregas muchas veces al día o tener una presencia web muy popular.
- En la implementación continua, con solo presionar un botón pasa por la construcción, las pruebas automatizadas, la automatización de los entornos y la producción. Esto se vuelve importante con respecto a las operaciones continuas.



© JMA 2020. All rights reserved

Automatización de la Prueba

- La automatización de la prueba permite ejecutar muchos casos de prueba de forma consistente y repetida en las diferentes versiones del sistema sujeto a prueba (SSP) y/o entornos. Pero la automatización de pruebas es más que un mecanismo para ejecutar un juego de pruebas sin interacción humana. Implica un proceso de diseño de productos de prueba, entre los que se incluyen:
 - Software.
 - Documentación.
 - Casos de prueba.
 - Entornos de prueba
 - Datos de prueba
- La Solución de Automatización Pruebas (SAP) debe permitir:
 - Implementar casos de prueba automatizados.
 - Monitorizar y controlar la ejecución de pruebas automatizadas.
 - Interpretar, informar y registrar los resultados de pruebas automatizadas.

© JMA 2020. All rights reserved

Objetivos de la automatización de pruebas

- Mejorar la eficiencia de la prueba.
- Aportar una cobertura de funciones más amplia.
- Reducir el coste total de la prueba.
- Realizar pruebas que los probadores manuales no pueden.
- Acortar el período de ejecución de la prueba.
- Aumentar la frecuencia de la prueba y reducir el tiempo necesario para los ciclos de prueba.

© JMA 2020. All rights reserved

Ventajas de la automatización de pruebas

- Se pueden realizar más pruebas por compilación ("build").
- La posibilidad de crear pruebas que no se pueden realizar manualmente (pruebas en tiempo real, remotas, en paralelo).
- Las pruebas pueden ser más complejas.
- Las pruebas se ejecutan más rápido.
- Las pruebas están menos sujetas a errores del operador.
- Uso más eficaz y eficiente de los recursos de pruebas
- Información de retorno más rápida sobre la calidad del software.
- Mejora de la fiabilidad del sistema (por ejemplo, repetibilidad, consistencia).
- Mejora de la consistencia de las pruebas.

© JMA 2020. All rights reserved

Desventajas de la automatización de pruebas

- Requiere tecnologías adicionales.
- Existencia de costes adicionales.
- Inversión inicial para el establecimiento de la SAP.
- Requiere un mantenimiento continuo de la SAP.
- El equipo necesita tener competencia en desarrollo y automatización.
- Puede distraer la atención respecto a los objetivos de la prueba, por ejemplo, centrándose en la automatización de casos de prueba a expensas del objetivo de las pruebas.
- Las pruebas pueden volverse más complejas.
- La automatización puede introducir errores adicionales.

© JMA 2020. All rights reserved

Limitaciones de la automatización de pruebas

- No todas las pruebas manuales se pueden automatizar.
- La automatización sólo puede comprobar resultados interpretables por la máquina.
- La automatización sólo puede comprobar los resultados reales que pueden ser verificados por un oráculo de prueba automatizado.
- No es un sustituto de las pruebas exploratorias.

© JMA 2020. All rights reserved

Entornos

- Un enfoque de varios entornos permite compilar, probar y liberar código con mayor velocidad y frecuencia para que la implementación sea lo más sencilla posible. Permite quitar la sobrecarga manual y el riesgo de una versión manual y, en su lugar, automatizar el desarrollo con un proceso de varias fases destinado a diferentes entorno.
 - Desarrollo: es donde se desarrollan los cambios en el software.
 - Prueba: permite que los evaluadores humanos o las pruebas automatizadas prueben el código nuevo y actualizado. Los desarrolladores deben aceptar código y configuraciones nuevos mediante la realización de pruebas unitarias en el entorno de desarrollo antes de permitir que esos elementos entren en uno o varios entornos de prueba.
 - Ensayo/preproducción: donde se realizan pruebas finales inmediatamente antes de la implementación en producción, debe reflejar un entorno de producción real con la mayor precisión posible.
 - UAT: Las pruebas de aceptación de usuario (UAT) permiten a los usuarios finales o a los clientes realizar pruebas para comprobar o aceptar el sistema de software antes de que una aplicación de software pueda pasar a su entorno de producción.
 - Producción: es el entorno con el que interactúan directamente los usuarios.

© JMA 2020. All rights reserved

Herramientas de Automatización

- Una cadena de herramientas de DevOps es una colección de herramientas que permite a los equipos de DevOps colaborar en todo el ciclo de vida del producto y abordar los aspectos básicos clave de DevOps. Las herramientas que incluye una cadena de herramientas de DevOps funcionan como una unidad integrada para la planificación, la integración continua, la entrega continua, las operaciones, la colaboración y los comentarios.
- Se puede adoptar diferentes tipos de cadenas de herramientas de DevOps:
 - Todo en uno: proporciona una solución completa que podría no integrarse con otras herramientas de terceros. Las cadenas de herramientas todo en uno pueden ser útiles para las organizaciones que empiezan su recorrido de DevOps.
 - Personalizado: permite a los equipos traer y mezclar herramientas existentes que conocen y usan ya en la cadena de herramientas de DevOps más amplia. La integración es fundamental para evitar que estos tipos de cadenas de herramientas pasen tiempo innecesario entre pantallas, inicien sesión en varios lugares y se enfrenten al reto de compartir información entre herramientas.

© JMA 2020. All rights reserved

Herramientas: Planificación

- Considere la posibilidad de adoptar una herramienta que admita prácticas de planeación continua:
 - Planificación de la versión
 - Identificación de características y epopeyas
 - Establecimiento de prioridades
 - Estimación
 - Definición de casos de usuario
 - Perfeccionamiento del trabajo pendiente
 - Planificación de un sprint
 - Scrum diario
 - Revisión de sprint
 - Retrospectiva

© JMA 2020. All rights reserved

Herramientas: Integración/entrega continua

- Al implementar Integración continua (CI)/Entrega continua (CD), considere la posibilidad de adoptar una herramienta que admita:
 - Sistemas de control de versiones. Todo el contenido del proyecto debe registrarse en un único repositorio de control de versiones como Git: código, pruebas, scripts de base de datos, scripts de compilación e implementación, y cualquier otra cosa necesaria para crear, instalar, ejecutar y probar la aplicación.
 - Estrategia de ramificación.
 - Compilaciones automatizadas.
 - Tenga en cuenta que su opción de repositorio también se ve influenciada por los requisitos de soberanía y residencia de datos (que deban hospedarse localmente en determinados países o regiones).
- Para minimizar la cantidad de configuración manual necesaria para aprovisionar recursos, considere la posibilidad de adoptar Infraestructura como código (IaC).
- Adopte herramientas de exploración de código para ayudar a detectar defectos de código lo antes posible. Incluya comprobaciones previas a la implementación para validar y confirmar los cambios antes de cualquier función de implementación (ejemplo: "what-if").
- Las herramientas de CI/CD aceleran el tiempo de comercialización del producto. Las herramientas que permiten parallelizar tareas y aprovechar la escalabilidad elástica en la infraestructura hospedada en la nube mejoran el rendimiento del proceso de CI/CD.
- Considere la posibilidad de usar las características de la herramienta de CI/CD que tomen métricas del rendimiento de DevOps. Los paneles e informes pueden realizar un seguimiento de los aspectos del proceso de desarrollo, como el plazo, el tiempo de ciclo, el progreso del trabajo, etc.

© JMA 2020. All rights reserved

Herramientas: Operaciones continuas

- Las operaciones continuas son un enfoque que ayuda a las organizaciones a mantener la continuidad del resultado entre los clientes y los sistemas internos a través de la entrega ininterrumpida de servicios o funciones críticos. Los objetivos de las operaciones continuas son:
 - Reducir o eliminar la necesidad de tiempos de inactividad planificados o interrupciones, como el mantenimiento programado, la optimización de la capacidad y la implementación.
 - Aumentar la confiabilidad y la resistencia generales de los sistemas en tres aspectos: personas, procesos y herramientas.
- Use herramientas nativas de nube para:
 - Supervisar las métricas clave para el rendimiento y la disponibilidad del servicio.
 - Obtener experiencia digital y Customer Insights.
 - Genere respuestas basadas en inteligencia para incidentes, la recuperación del sistema o el escalado.
 - Automatice el mantenimiento proactivo y tareas como la implementación o las actualizaciones del sistema.

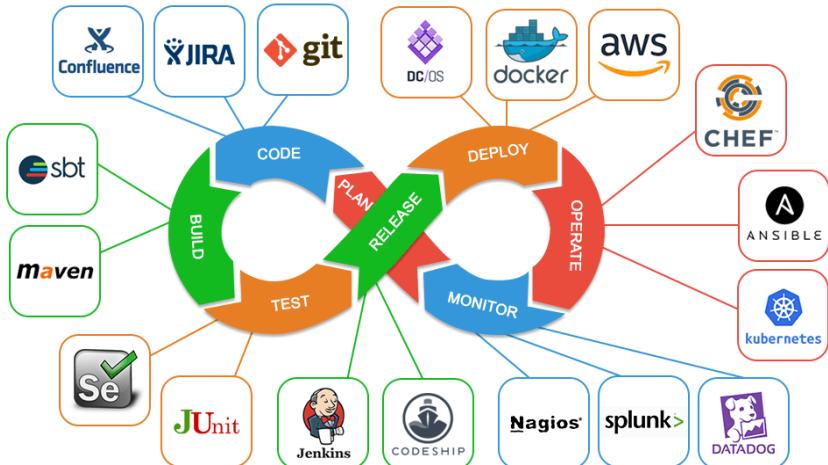
© JMA 2020. All rights reserved

Herramientas: Colaboración y comentarios

- Los bucles de comentarios rápidos constituyen el núcleo del proceso de CI/CD. Una herramienta de CI/CD usa comentarios para resolver condiciones en la lógica de flujo de trabajo de CI/CD y muestra información a los usuarios, normalmente a través de un panel.
- La compatibilidad con las notificaciones por correo electrónico y la integración con IDE o plataformas de comunicación garantiza que pueda mantenerse informado sobre lo que sucede sin tener que comprobar un panel. Asegúrese de disponer de la opción de configurar las alertas que va a recibir, ya que obtener demasiadas alertas hace que se transformen en ruido de fondo (spam).
- Cualquier herramienta que elija para la colaboración debe admitir las siguientes prácticas de colaboración:
 - Colaboración Kanban
 - Colaboración de contenido wiki
 - Colaboración ChatOps
 - Centro de reunión

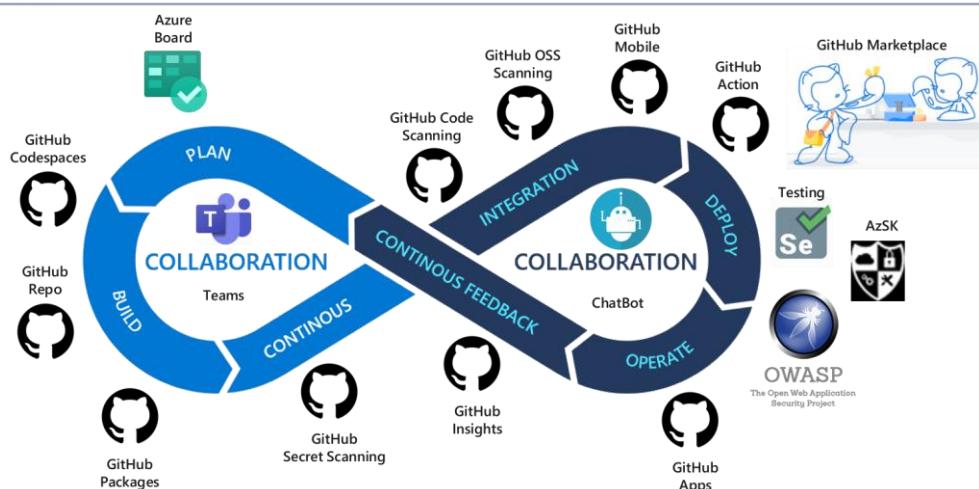
© JMA 2020. All rights reserved

Herramientas de Automatización



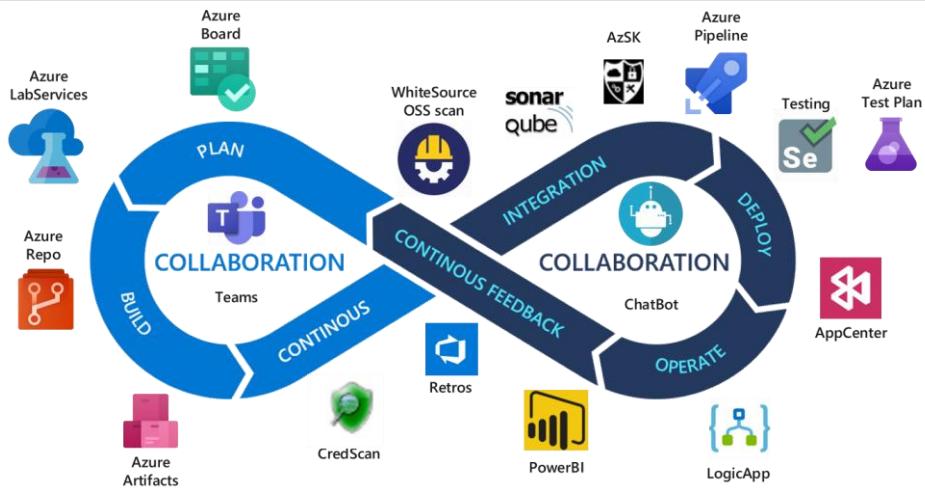
© JMA 2020. All rights reserved

GitHub



© JMA 2020. All rights reserved

Azure DevOps



© JMA 2020. All rights reserved

Azure DevOps

- Los puntos del esquema anterior exponen cómo sería un proceso de gestión del ciclo de vida de la una aplicación:
 1. Podemos usar nuestro Visual Studio para desarrollar nuestro código, aunque no es necesario usar esta herramienta en particular, podemos usar cualquier herramienta de desarrollo de código.
 2. De manera regular podremos ir guardando nuestro código en nuestro repositorio de Git dentro del módulo de Azure Repos.
 3. Mediante la integración continua, podemos desencadenar pruebas y la compilación de la aplicación mediante el módulo Azure Test Plans.
 4. Tras esto, con la implementación automatizada de los artefactos y la aplicación de valores de configuración específicos por entornos, podremos realizar despliegues a cada uno de los entornos que compongan nuestra infraestructura con los Azure Pipelines.
 5. En el caso de que no dispongamos de Azure Web Apps, los artefactos anteriormente mencionados también se pueden desplegar a nuestros servidores locales.
 6. Azure Application Insights es otra herramienta propia de Application Performance Management (APM) extensible para desarrolladores web en varias plataformas. Que pueden usar para supervisar y administrar la información sobre el estado, el rendimiento y el uso.
 7. Una vez analizados los resultados podemos volver al código.
 8. Mediante Azure Boards podremos hacer un seguimiento del trabajo diario y pendiente de los equipos

© JMA 2020. All rights reserved

Prácticas de DevOps

- Más allá del establecimiento de una cultura de DevOps, los equipos ponen en práctica el método DevOps implementando determinadas prácticas a lo largo del ciclo de vida de las aplicaciones.
- Algunas de estas prácticas ayudan a agilizar, automatizar y mejorar una fase específica.
- Otras abarcan varias fases y ayudan a los equipos a crear procesos homogéneos que favorezcan la productividad.
- Estas prácticas incluyen:
 - Control de versiones
 - Desarrollo ágil de software
 - Integración y entrega continuas (CI/CD)
 - Infraestructura como código
 - Administración de configuración
 - Supervisión continua

© JMA 2020. All rights reserved

Control de versiones

- Control de versiones es la práctica de administrar el código y la documentación por versiones, haciendo un seguimiento de las revisiones y del historial de cambios para facilitar la revisión y la recuperación del código.
- Esta práctica suele implementarse con sistemas de control de versiones, como Git, que permite que varios desarrolladores colaboren para crear código. Estos sistemas proporcionan un proceso claro para fusionar mediante combinación los cambios en el código que tienen lugar en los mismos archivos, controlar los conflictos y revertir los cambios a estados anteriores.
- El uso del control de versiones es una práctica de DevOps fundamental que ayuda a los equipos de desarrollo a trabajar juntos, dividir las tareas de programación entre los miembros del equipo y almacenar todo el código para poder recuperarlo fácilmente si fuese necesario.
- El control de versiones es también un elemento necesario en otras prácticas, como la integración continua y la infraestructura como código.

© JMA 2020. All rights reserved

SEMVER

- El sistema SEMVER (Semantic Versioning) es un conjunto de reglas para proporcionar un significado claro y definido a las versiones de los proyectos.
- El sistema SEMVER se compone de 3 números, siguiendo la estructura X.Y.Z:
 - X se denomina Major: indica cambios rupturistas
 - Y se denomina Minor: indica cambios compatibles con la versión anterior
 - Z se denomina Patch: indica resoluciones de bugs (compatibles)
- Básicamente, cuando se arregla un bug se incrementa el patch, cuando se introduce una mejora se incrementa el minor y cuando se introducen cambios que no son compatibles con la versión anterior, se incrementa el major.
- De este modo cualquier desarrollador sabe qué esperar ante una actualización de su librería favorita. Si sale una actualización donde el major se ha incrementado, sabe que tendrá que ensuciarse las manos con el código para pasar su aplicación existente a la nueva versión.

© JMA 2020. All rights reserved

Políticas de versionado

- Dentro de la política de versionado es conveniente planificar la obsolescencia y la política de desaprobación.
- La obsolescencia programada establece el periodo máximo, como una franja temporal o un número de versiones, en que se va a dar soporte a cada versión, evitando los sobrecostes derivados de mantener versiones obsoletas indefinidamente.
- Dentro de la política de desaprobación, para ayudar a garantizar que los consumidores tengan tiempo suficiente y una ruta clara de actualización, se debe establecer el número de versiones en que se mantendrá una característica marcada como obsoleta antes de su desaparición definitiva.
- La obsolescencia programada y la política de desaprobación beneficia a los consumidores del producto porque proporcionan estabilidad y sabrán qué esperar a medida que los productos evolucionen.
- Para mejorar la calidad y avanzar las novedades, se podrán realizar lanzamientos de versiones Beta y Release Candidates (RC) o revisiones para cada versión mayor y menor. Estas versiones provisionales desaparecerán con el lanzamiento de la versión definitiva.

© JMA 2020. All rights reserved

Desarrollo ágil de software

- El método ágil es un enfoque de desarrollo de software que hace hincapié en la colaboración en equipo, en los comentarios de los clientes y usuarios, y en una gran capacidad de adaptación a los cambios mediante ciclos cortos de lanzamiento de versiones.
- Los equipos que practican la metodología ágil proporcionan mejoras y cambios continuos a los clientes, recopilan sus comentarios y, después, aprenden y ajustan el software en función de lo que el cliente quiere y necesita.
- El método ágil es muy diferente a otros marcos más tradicionales, como el modelo en cascada, que incluye ciclos de lanzamiento de versiones largos definidos por fases secuenciales. Kanban y Scrum son dos marcos populares asociados al método ágil.

© JMA 2020. All rights reserved

Infraestructura como código

- La infraestructura como código define las topologías y los recursos del sistema de un modo descriptivo que permite a los equipos administrar esos recursos igual que lo harían con el código. Las diferentes versiones de esas definiciones se pueden almacenar en sistemas de control de versiones, donde se pueden revisar y revertir, de nuevo, igual que el código.
- La práctica de la infraestructura como código permite a los equipos implementar recursos del sistema de un modo confiable, repetible y controlado. Además, la infraestructura como código ayuda a automatizar la implementación y reduce el riesgo de errores humanos, especialmente en entornos complejos de gran tamaño.
- Esta solución repetible y confiable para la implementación de entornos permite a los equipos mantener entornos de desarrollo y pruebas que sean idénticos al entorno de producción. De igual modo, la duplicación de entornos en otros centros de datos y en plataformas en la nube es más sencilla y más eficiente.

© JMA 2020. All rights reserved

Administración de configuración

- Administración de la configuración hace referencia a la administración del estado de los recursos de un sistema, incluidos los servidores, las máquinas virtuales y las bases de datos.
- El uso de herramientas de administración de la configuración permite a los equipos distribuir cambios de un modo controlado y sistemático, lo que reduce el riesgo de modificar la configuración del sistema. Los equipos utilizan herramientas de administración de la configuración para hacer un seguimiento del estado del sistema y evitar alteraciones en la configuración, que es como se desvía la configuración de un recurso del sistema a lo largo del tiempo del estado definido para él.
- Al usarla junto con la infraestructura como código, resulta fácil elaborar plantillas y automatizar la definición y la configuración de sistemas, lo que permite a los equipos usar entornos complejos a escala.

© JMA 2020. All rights reserved

Supervisión continua

- Supervisión continua significa tener visibilidad total y en tiempo real del rendimiento y el estado de toda la pila de aplicaciones, desde la infraestructura subyacente donde se ejecutan las aplicaciones hasta los componentes de software de niveles superiores.
- La visibilidad consiste en la recopilación de datos de telemetría y metadatos, así como en el establecimiento de alertas para condiciones predefinidas que garanticen la atención de un operador. La telemetría incluye registros y datos de eventos recopilados de varias partes del sistema que se almacenan donde pueden analizarse y consultarse.
- Los equipos de DevOps de alto rendimiento se aseguran de establecer alertas útiles que les permitan tomar medidas y recopilan datos de telemetría muy completos para obtener conclusiones a partir de enormes cantidades de datos.
- Estas conclusiones ayudan a los equipos a mitigar los problemas en tiempo real y a ver cómo mejorar la aplicación en futuros ciclos de desarrollo.

© JMA 2020. All rights reserved

DevSecOps

© JMA 2020. All rights reserved

DevSecOps

- DevSecOps significa desarrollo, seguridad y operaciones. Se trata de un enfoque que aborda la cultura, la automatización y el diseño de plataformas, e integra la seguridad como una responsabilidad compartida durante todo el ciclo de vida.
- DevOps no solo concierne a los equipos de desarrollo y operaciones. Si desea aprovechar al máximo la agilidad y la capacidad de respuesta de los enfoques de DevOps, la seguridad de la TI también debe desempeñar un papel integrado en el ciclo de vida completo de sus aplicaciones.
- Antes, el papel de la seguridad estaba aislado y a cargo de un equipo específico en la etapa final del desarrollo. Cuando los ciclos de desarrollo duraban meses o incluso años, no pasaba nada. Pero eso quedó en el pasado. Una metodología efectiva de DevOps garantiza ciclos de desarrollo rápidos y frecuentes (a veces de semanas o días), pero las prácticas de seguridad obsoletas pueden revertir incluso las iniciativas de DevOps más eficientes.



© JMA 2020. All rights reserved

Desafíos

- **Los equipos son reacios a integrarse:** La esencia de DevSecOps es la integración de equipos para que puedan trabajar juntos en lugar de forma independiente. Sin embargo, no todo el mundo está preparado para hacer el cambio porque ya está acostumbrado a los procesos de desarrollo actuales.
- **Guerra de herramientas:** Dado que los equipos han estado trabajando por separado, han estado utilizando diferentes métricas y herramientas. En consecuencia, les resulta difícil ponerse de acuerdo sobre dónde tiene sentido integrar las herramientas y dónde no. No es fácil reunir herramientas de varios departamentos e integrarlas en una plataforma. Por lo tanto, el desafío es seleccionar las herramientas adecuadas e integrarlas adecuadamente para construir, implementar y probar el software de manera continua.
- **Implementar seguridad en IC/DC:** Generalmente, se ha pensado en la seguridad como algo que llega al final del ciclo de desarrollo. Sin embargo, con DevSecOps, la seguridad es parte de la integración y el desarrollo continuos (IC/DC). Para que DevSecOps tenga éxito, los equipos no pueden esperar que los procesos y herramientas de DevOps se adapten a los viejos métodos de seguridad. Al integrar los controles de seguridad en DevOps, las organizaciones están adoptando el nuevo modelo DevSecOps para aprovechar todo el potencial de IC/DC. Cuando las empresas implementan tecnologías de control de acceso o seguridad desde el principio, se aseguran de que esos controles estén en línea con un flujo de IC/DC.

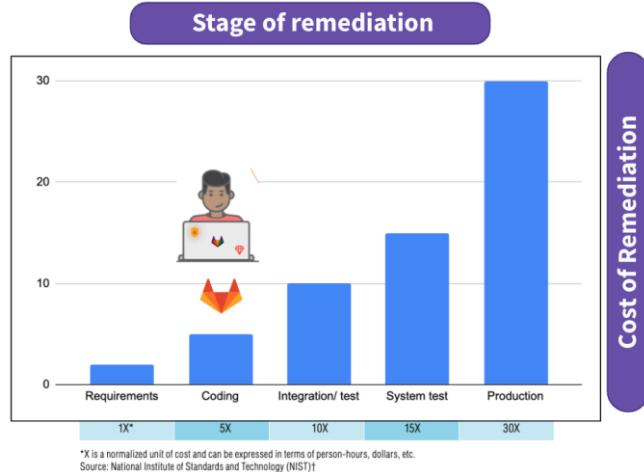
© JMA 2020. All rights reserved

Beneficios

- Desarrollar software seguro desde el diseño.
- Identificación temprana de vulnerabilidades en el código y ataques.
- Aplicar la seguridad de forma más rápida y ágil.
- Responder a los cambios y requisitos rápidamente.
- Mejorar la colaboración y comunicación entre equipos.
- Generar una mayor conciencia sobre seguridad entre todos los miembros.
- Enfocarse en generar el mayor valor de negocio.

© JMA 2020. All rights reserved

Detección temprana



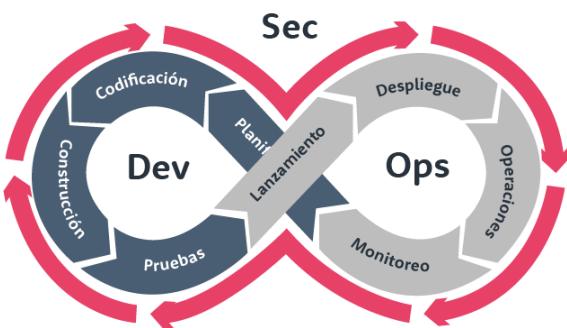
© JMA 2020. All rights reserved

Implementación

- DevSecOps supone un cambio en la filosofía de trabajo, donde la seguridad se convierte en una responsabilidad compartida (extendida a los equipos de desarrollo y operaciones) e integrada a lo largo de todo el proceso de desarrollo y despliegue. Para poner en práctica DevSecOps, la seguridad debe convertirse en una condición más en el proceso de diseño, desarrollo y entrega de software.
- DevSecOps consta de dos partes diferenciadas pero que forman parte del mismo proceso.
 - Seguridad como código (SaC): cuando se incorporan la seguridad al desarrollo y las operaciones de herramientas informáticas o aplicaciones.
 - Infraestructura como código (IaC): hace referencia al conjunto de herramientas de desarrollo de operaciones (DevOps) utilizadas para configurar y actualizar los componentes de la infraestructura para garantizar un entorno controlado. Trata las infraestructuras como un software.

© JMA 2020. All rights reserved

Ciclo de vida



- Asesorar a los desarrolladores sobre las vulnerabilidades de seguridad para ayudar a configurar las herramientas y las opciones de diseño.
- Trabajar con el equipo de operaciones para llevar a cabo un mantenimiento de seguridad regular y actualizaciones de la infraestructura.
- Implementar pruebas de seguridad automatizadas para la arquitectura, diseño y despliegue en cada proyecto nuevo.
- Usar técnicas de monitorización continua para asegurar que los sistemas de seguridad están trabajando según lo previsto.

© JMA 2020. All rights reserved

Buenas prácticas

- **Implemente la automatización para proteger el entorno de IC/DC:** Uno de los aspectos clave del entorno IC/DC es la velocidad. Y eso significa que la automatización es necesaria para integrar la seguridad en este entorno, al igual que incorporar los controles y pruebas de seguridad esenciales a lo largo del ciclo de vida del desarrollo. También es importante implementar pruebas de seguridad automatizadas en las canalizaciones de IC/DC para permitir el análisis de vulnerabilidades en tiempo real.
- **Aborde los problemas de seguridad de la tecnología de código abierto:** Está aumentando el uso de herramientas de código abierto para el desarrollo de aplicaciones. Por lo tanto, las organizaciones deben abordar las preocupaciones de seguridad relacionadas con el uso de dichas tecnologías. Sin embargo, dado que los desarrolladores están demasiado ocupados para revisar el código fuente abierto, es importante implementar procesos automatizados para administrar el código fuente abierto, así como otras herramientas y tecnologías de terceros. Por ejemplo, utilidades como Open Web Application Security Project (OWASP) pueden verificar que no haya vulnerabilidades en el código que dependa de componentes de código abierto.
- **Integre el sistema de seguridad de la aplicación con el sistema de gestión de tareas:** Esto creará automáticamente una lista de tareas con errores que el equipo de seguridad de la información puede ejecutar. Además, proporcionará detalles procesables, incluida la naturaleza del defecto, su gravedad y la mitigación necesaria. Como tal, el equipo de seguridad puede solucionar problemas antes de que terminen en los entornos de desarrollo y producción.

© JMA 2020. All rights reserved

Seguridad del entorno y de los datos

- **Estandarice y automatice el entorno:** los servicios deben tener la menor cantidad de privilegios posible para reducir las conexiones y los accesos no autorizados.
- **Centralice las funciones de control de acceso y de identidad de los usuarios:** el control de acceso estricto y los mecanismos de autenticación centralizados son fundamentales para proteger los microservicios, ya que la autenticación se inicia en varios puntos.
- **Aíslle de la red y entre sí aquellos contenedores que ejecutan microservicios:** abarca tanto los datos en tránsito como en reposo, ya que ambos pueden ser objetivos valiosos para los atacantes.
- **Cifre los datos entre las aplicaciones y los servicios:** una plataforma de organización de contenedores con funciones de seguridad integradas disminuye las posibilidades de accesos no autorizados.
- **Incorpore puertas de enlace de API seguras:** las API seguras aumentan el control de los enrutamientos y las autorizaciones. Al disminuir la cantidad de API expuestas, las empresas pueden reducir las superficies de ataque.

© JMA 2020. All rights reserved

Seguridad del proceso de CI/CD

- **Integre análisis de seguridad para los contenedores:** estos deberían ser parte del proceso para agregar contenedores al registro.
- **Automatice las pruebas de seguridad en el proceso de integración continua:** esto incluye ejecutar herramientas de análisis de seguridad estática como parte de las compilaciones, así como examinar las imágenes de contenedores prediseñadas para detectar puntos vulnerables de seguridad conocidos a medida que se incorporan al canal de diseño.
- **Incorpore pruebas automatizadas para las funciones de seguridad al proceso de pruebas de aceptación:** automatice las pruebas de validación de los datos de entrada, así como las funciones de verificación, autenticación y autorización.
- **Automatice las actualizaciones de seguridad, como la aplicación de parches para los puntos vulnerables conocidos:** lleve a cabo este proceso mediante el canal de DevOps. Esto elimina la necesidad de que los administradores inicien sesión en los sistemas de producción mientras crean un registro de cambios documentado y rastreable.
- **Automatice las funciones de gestión de la configuración de los sistemas y los servicios:** de esta manera, podrá cumplir con las políticas de seguridad y eliminar los errores manuales. También se recomienda automatizar los procesos de auditoría y corrección.

© JMA 2020. All rights reserved

Integración en el ciclo de vida

- Pruebas estáticas de seguridad del software
 - Escanea el código fuente de la aplicación y los archivos binarios para detectar posibles vulnerabilidades.
- Pruebas dinámicas de seguridad del software
 - Escanea la aplicación en tiempo de ejecución para detectar vulnerabilidades como:
 - SQL injection
 - DoS attack
 - XSS
 - Es necesario que la aplicación se este ejecutando en un entorno de laboratorio, donde se realizan las pruebas de afuera hacia adentro sin tener acceso al código fuente. También hacen parte de los tipos de pruebas conocidas como de caja negra.
- Escaneo de dependencias
 - Analiza las dependencias externas del proyecto como npm, ruby gem, composer, entre otras para detectar vulnerabilidades en cada commit de código. Por ejemplo, verificar que no estamos utilizando librerías de código obsoletas o deprecadas.
- Cumplimiento de licencias
 - Busca automáticamente en las dependencias del proyecto las licencias aprobadas o en lista negra de su organización para evitar utilizar software pirata.

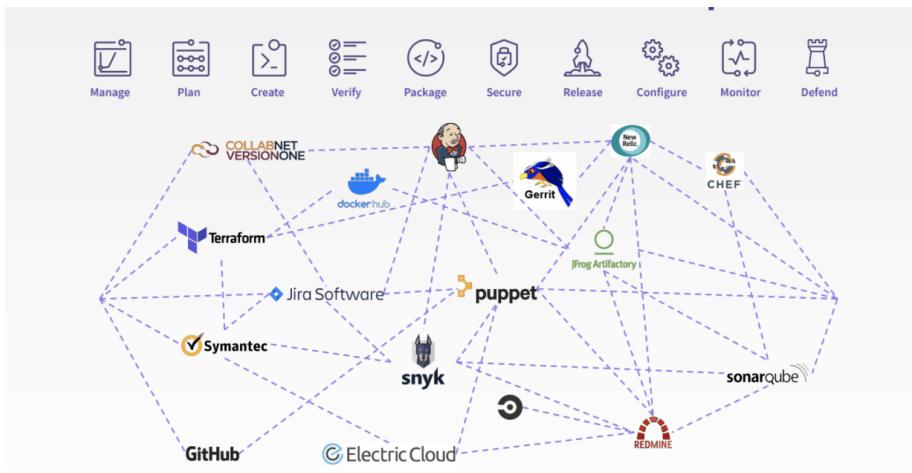
© JMA 2020. All rights reserved

¿Es DevSecOps adecuado para tu equipo?

- Los beneficios de DevSecOps son claros: velocidad, eficiencia y colaboración. Pero, ¿cómo saber si es adecuado para tu equipo? Si tu organización se enfrenta alguno de los siguientes desafíos, un enfoque de DevSecOps podría ser una buena medida:
 - Los equipos de desarrollo, seguridad y operaciones están aislados. Si el desarrollo y las operaciones están aislados de los problemas de seguridad, no pueden crear software seguro. Y si los equipos de seguridad no forman parte del proceso de desarrollo, no pueden identificar los riesgos de manera proactiva. DevSecOps reúne a los equipos para mejorar los flujos de trabajo y compartir ideas. Las organizaciones podrían incluso ver una mejora en la moral y la retención de los empleados.
 - Los largos ciclos de desarrollo dificultan satisfacer las demandas de los clientes o de las partes interesadas. Una de las razones de la dificultad podría ser la seguridad. DevSecOps implementa la seguridad en cada paso del ciclo de vida del desarrollo, lo que significa que una seguridad sólida no requiere que todo el proceso se detenga.
 - Se está migrando a la nube (o considerando hacerlo). Mudarse a la nube a menudo significa incorporar nuevos procesos, herramientas y sistemas de desarrollo. Es el momento perfecto para hacer que los procesos sean más rápidos y seguros, y DevSecOps podría hacerlo mucho más fácil.

© JMA 2020. All rights reserved

Herramientas



© JMA 2020. All rights reserved

Referencias

- OWASP Top Ten
 - <https://owasp.org/www-project-top-ten/>
- OWASP Top 10 CI/CD Security Risks
 - <https://owasp.org/www-project-top-10-ci-cd-security-risks/>
- GitLab: Global DevSecOps Survey
 - <https://about.gitlab.com/developer-survey/>
- <https://www.devsecops.org/>

© JMA 2020. All rights reserved

INTRODUCCIÓN A LAS TÉCNICAS DE PRUEBAS

© JMA 2020. All rights reserved

Introducción

- El software generado en la fase de implementación no puede ser "entregado" al cliente para que lo utilice, sin practicarle antes una serie de pruebas.
- La fase de pruebas tienen como objetivo encontrar defectos en el sistema final debido a la omisión o mala interpretación de alguna parte del análisis o el diseño. Los defectos deberán entonces detectarse y corregirse en esta fase del proyecto.
- En ocasiones los defectos pueden deberse a errores en la implementación de código (errores propios del lenguaje o sistema de implementación), aunque en esta etapa es posible realizar una efectiva detección de los mismos, estos deben ser detectados y corregidos en la fase de implementación.
- La prueba puede ser llevada a cabo durante la implementación, para verificar que el software se comporta como su diseñador pretendía, y después de que la implementación esté completa.

© JMA 2020. All rights reserved

Introducción

- Esta fase tardía de prueba comprueba la conformidad con los requerimientos y asegura la fiabilidad del sistema.
- Las distintas clases de prueba utilizan **clases de datos de prueba** diferentes:
 - La **prueba estadística**.
 - La **prueba de defectos**.
 - La **prueba de regresión**.

© JMA 2020. All rights reserved

Prueba estadística

- La **prueba estadística** se puede utilizar para probar el rendimiento del programa y su confiabilidad.
- Las pruebas se diseñan para reflejar la frecuencia de entradas reales de usuario.
- Después de ejecutar las pruebas, se puede hacer una estimación de la confiabilidad operacional del sistema.
- El rendimiento del programa se puede juzgar midiendo la ejecución de las pruebas estadísticas.

© JMA 2020. All rights reserved

Prueba de defectos

- La **prueba de defectos** intenta incluir áreas donde el programa no está de acuerdo con sus especificaciones.
- Las pruebas se diseñan para revelar la presencia de defectos en el sistema.
- Cuando se han encontrado defectos en un programa, éstos deben ser localizados y eliminados. A este proceso se le denomina **depuración**.
- La prueba de defectos y la depuración son consideradas a veces como parte del mismo proceso. En realidad, son muy diferentes, puesto que la prueba establece la existencia de errores, mientras que la depuración se refiere a la localización y corrección de estos errores.

© JMA 2020. All rights reserved

Error, defecto o fallo

- En el área del aseguramiento de la calidad del software, debemos tener claros los conceptos de Error, Defecto y Fallo.
- En muchos casos se utilizan indistintamente pero representan conceptos diferentes:
 - Error: Es una acción humana, una idea equivocada de algo, que produce un resultado incorrecto. El error es una equivocación por parte del desarrollador o del analista.
 - Defecto: Es una imperfección de un componente causado por un error. El defecto se encuentra en algún componente del sistema. El analista de pruebas es quien debe encontrar el defecto ya que es el encargado de elaborar y ejecutar los casos de prueba.
 - Fallo: Es la manifestación visible de un defecto. Es decir que si un defecto es encontrado durante la ejecución de una aplicación entonces va a producir un fallo.
- Un error puede generar uno o más defectos y un defecto puede causar un fallo.

© JMA 2020. All rights reserved

Prueba de regresión

- Durante la depuración se debe generar hipótesis sobre el comportamiento observable del programa para probar entonces estas hipótesis esperando provocar un fallo y encontrar el defecto que causó la anomalía en la salida.
- Después de descubrir un error en el programa, debe corregirse y volver a probar el sistema.
- A esta forma de prueba se le denomina **prueba de regresión**.
- La prueba de regresión se utiliza para comprobar que los cambios hechos a un programa no han generado nuevos fallos en el sistema.

© JMA 2020. All rights reserved

Conflictos de intereses

- Aparte de esto, en cualquier proyecto software existe un conflicto de intereses inherente que aparece cuando comienza la prueba:
 - Desde un punto de vista psicológico, el análisis, diseño y codificación del software son tareas constructivas.
 - El ingeniero de software crea algo de lo que está orgulloso, y se enfrenta a cualquiera que intente sacarle defectos.
 - La prueba, entonces, puede aparecer como un intento de "romper" lo que ha construido el ingeniero de software.
 - Desde el punto de vista del constructor, la prueba se puede considerar (psicológicamente) destructiva.
 - Por tanto, el constructor anda con cuidado, diseñando y ejecutando pruebas que demuestren que el programa funciona, en lugar de detectar errores.
 - Desgraciadamente los errores seguirán estando, y si el ingeniero de software no los encuentra, lo hará el cliente.

© JMA 2020. All rights reserved

Desarrollador como probador

- A menudo, existen ciertos **malentendidos** que se pueden deducir equivocadamente de la anterior discusión:
 1. El desarrollador del software no debe entrar en el proceso de prueba.
 2. El software debe ser “puesto a salvo” de extraños que puedan probarlo de forma despiadada.
 3. Los encargados de la prueba sólo aparecen en el proyecto cuando comienzan las etapas de prueba.
- Cada una de estas frases es incorrecta.
- El **desarrollador** siempre es responsable de probar las unidades individuales (módulos) del programa, asegurándose de que cada una lleva a cabo la función para la que fue diseñada.
- En muchos casos, también se encargará de la prueba de integración de todos los elementos en la estructura total del sistema.

© JMA 2020. All rights reserved

Grupo independiente de prueba

- Sólo una vez que la arquitectura del software esté completa, entra en juego un **grupo independiente de prueba**, que debe eliminar los problemas inherentes asociados con el hecho de permitir al constructor que pruebe lo que ha construido. Una prueba independiente elimina el conflicto de intereses que, de otro modo, estaría presente.
- En cualquier caso, el desarrollador y el grupo independiente deben trabajar estrechamente a lo largo del proyecto de software para asegurar que se realizan pruebas exhaustivas.
- Mientras se dirige la prueba, el desarrollador debe estar disponible para corregir los errores que se van descubriendo.

© JMA 2020. All rights reserved

Principios fundamentales

- Hay 6 principios fundamentales respecto a las metodologías de pruebas que deben quedar claros desde el primer momento aunque volveremos a ellos continuamente:
 - Las pruebas exhaustivas no son viables
 - Ejecución de pruebas bajo diferentes condiciones
 - El proceso de pruebas no puede demostrar la ausencia de defectos
 - Las pruebas no garantizan ni mejoran la calidad del software
 - Las pruebas tienen un coste
 - Inicio temprano de pruebas

© JMA 2020. All rights reserved

Las pruebas exhaustivas no son viables

- Es imposible, inviable, crear casos de prueba que cubran todas las posibles combinaciones de entrada y salida que pueden llegar a tener las funcionalidades (salvo que sean triviales).
- Por otro lado, en proyectos cuyo número de casos de uso o historias de usuario desarrollados sea considerable, se requeriría de una inversión muy alta en cuanto a recursos y tiempo necesarios para cubrir con pruebas todas las funcionalidades del sistema.
- Por lo tanto es conveniente realizar un análisis de riesgos de todas las funcionalidades y determinar en este punto cuales serán objeto de prueba y cuales no, creando pruebas que cubran el mayor número de casos de prueba posibles.

© JMA 2020. All rights reserved

Ejecución de pruebas bajo diferentes condiciones

- El plan de pruebas determina la condiciones y el número de ciclos de prueba que se ejecutarán sobre las funcionalidades del negocio.
- Por cada ciclo de prueba, se generan diferentes tipos de condiciones, basados principalmente en la variabilidad de los datos de entrada y en los conjuntos de datos utilizados.
- No es conveniente, ejecutar en cada ciclo, los casos de prueba basados en los mismos datos del ciclo anterior, dado que con mucha probabilidad, se obtendrán los mismos resultados.
- Ejecutar ciclos bajo diferentes tipos de condiciones, permitirá identificar posibles fallos en el sistema que antes no se detectaron y no son fácilmente reproducibles.

© JMA 2020. All rights reserved

El proceso no puede demostrar la ausencia de defectos

- Independientemente de la rigurosidad con la que se haya planeado el proceso de pruebas de un producto, nunca será posible garantizar al ejecutar este proceso, la ausencia total de defectos (es inviable una cobertura del 100%).
- Una prueba se considera un éxito si detecta un error. Si no detecta un error no significa que no haya error, significa que no se ha detectado.
- Un proceso de pruebas riguroso puede garantizar una reducción significativa de los posibles fallos y/o defectos del software, pero nunca podrá garantizar que el software no fallará en producción.

© JMA 2020. All rights reserved

Las pruebas no garantizan ni mejoran la calidad del software

- Las pruebas ayudan a **mejorar la percepción** de la calidad permitiendo la eliminación de los defectos detectados.
- La calidad del software viene determinada por las metodologías y buenas prácticas empleadas en el desarrollo del mismo.
- Las pruebas **permiten medir la calidad** del software, lo que permite, a su vez, mejorar los procesos de desarrollo que son los que conllevan la mejora de la calidad y permiten garantizar un nivel determinado de calidad.

© JMA 2020. All rights reserved

Las pruebas tienen un coste

- Aunque exige dedicar esfuerzo (coste para las empresas) para crear y mantener los test, los beneficios obtenidos son mayores que la inversión realizada.
- La creciente inclusión del software como un elemento más de muchos sistemas productivos y la importancia de los "costes" asociados a un fallo del mismo están motivando la creación de pruebas minuciosas y bien planificadas.
- No es raro que una organización de desarrollo de software gaste el 40 por 100 del esfuerzo total de un proyecto en la prueba.
- En casos extremos, la prueba del software para actividades críticas (por ejemplo, control de tráfico aéreo, o control de reactores nucleares) puede costar ¡de 3 a 5 veces más que el resto de los pasos de la ingeniería del software juntos!
- El coste de hacer las pruebas es siempre inferior al coste de no hacer las pruebas (deuda técnica).

© JMA 2020. All rights reserved

Inicio temprano de pruebas

- Si bien la fase de pruebas es la última del ciclo de vida, las actividades del proceso de pruebas deben ser incorporadas desde la fase de especificación, incluso antes de que se ejecutan las etapas de análisis y diseño.
- De esta forma los documentos de especificación y de diseño deben ser sometidos a revisiones y validaciones, lo que ayudará a detectar problemas en la lógica del negocio mucho antes de que se escriba una sola línea de código.
- Cuanto mas temprano se detecte un defecto, ya sea sobre los entregables de especificación, diseño o sobre el producto, menor impacto tendrá en el desarrollo y menor será el costo de dar solución a dichos defectos.

© JMA 2020. All rights reserved

EL PROCESO DE PRUEBAS

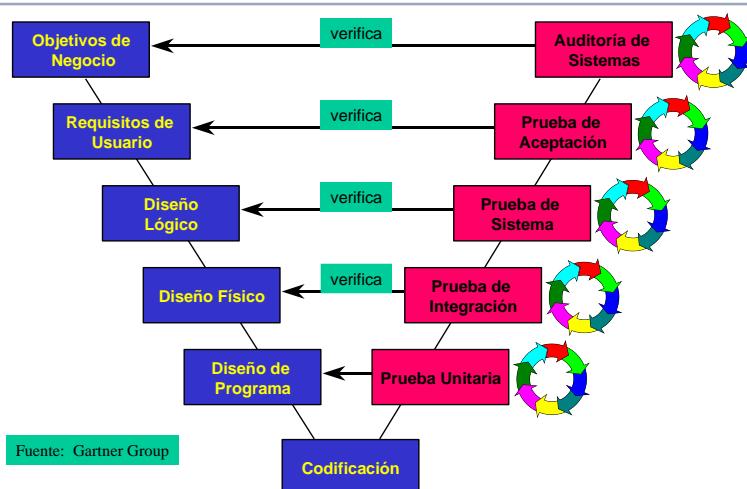
© JMA 2020. All rights reserved

El proceso

- Excepto para programas pequeños, los sistemas no deberían probarse como un único elemento indivisible.
- Los sistemas grandes se construyen a partir de subsistemas que se construyen a partir de módulos, compuestos de funciones y procedimientos.
- El proceso de prueba debería trabajar por etapas, llevando a cabo la prueba de forma incremental a la vez que se realiza la implementación del sistema, siguiendo el modelo en V.

© JMA 2020. All rights reserved

Proceso de pruebas: Ciclo en V



© JMA 2020. All rights reserved

Ciclo en V

- El modelo en V establece una simetría entre las fases de desarrollo y las pruebas.
- Las principales consideraciones se basan en la inclusión de las actividades de planificación y ejecución de pruebas como parte del proyecto de desarrollo.
- Inicialmente, la ingeniería del sistema define el papel del software y conduce al análisis de los requisitos del software, donde se establece el campo de información, la función, el comportamiento, el rendimiento, las restricciones y los criterios de validación del software. Al movernos hacia abajo, llegamos al diseño y, por último, a la codificación, el vértice de la V.
- Para desarrollar las pruebas seguimos el camino ascendente por la otra rama.
- Partiendo de los elementos más básicos, probamos que funcionan como deben (lo que hacen, lo hacen bien). Los combinamos y probamos que siguen funcionando como deben. Para terminar probamos que hacen lo que deben (que hacen todo lo que tienen que hacer).

© JMA 2020. All rights reserved

Desarrollo iterativo e incremental

- El desarrollo incremental implica establecer requisitos, diseñar, construir y probar un sistema en fragmentos, lo que significa que las prestaciones del software crecen de forma incremental. El tamaño de estos incrementos de prestaciones varía, ya que algunos métodos tienen piezas más grandes y otros más pequeñas. Los incrementos de prestaciones pueden ser tan pequeños como un simple cambio en una pantalla de la interfaz de usuario o una nueva opción en una consulta.
- El desarrollo iterativo se produce cuando se especifican, diseñan, construyen y prueban conjuntamente grupos de prestaciones en una serie de ciclos, a menudo, de una duración fija. Las iteraciones pueden implicar cambios en las prestaciones desarrolladas en iteraciones anteriores, junto con cambios en el alcance del proyecto. Cada iteración proporciona software operativo, que es un subconjunto creciente del conjunto general de prestaciones hasta que se entrega el software final o se detiene el desarrollo.

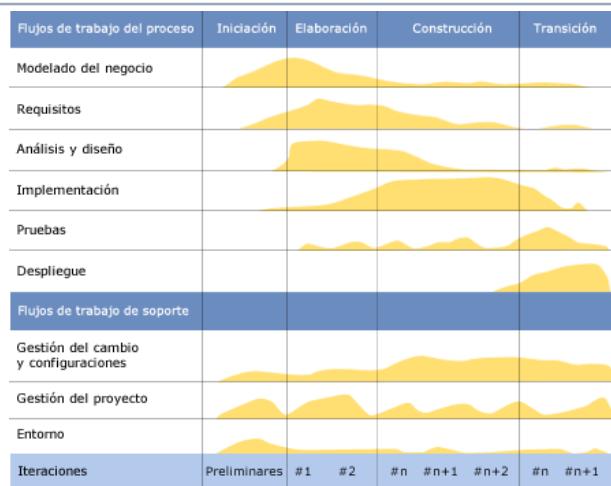
© JMA 2020. All rights reserved

Desarrollo iterativo e incremental

- Rational Unified Process (RUP):
 - Cada iteración tiende a ser relativamente larga (por ejemplo, de dos a tres meses), y los incrementos de las prestaciones son proporcionalmente grandes, como por ejemplo dos o tres grupos de prestaciones relacionadas.
- Scrum:
 - Cada iteración tiende a ser relativamente corta (por ejemplo, horas, días o unas pocas semanas), y los incrementos de las prestaciones son proporcionalmente pequeños, como unas pocas mejoras y/o dos o tres prestaciones nuevas.
- Kanban:
 - Implementado con o sin iteraciones de longitud fija, que puede ofrecer una sola mejora o prestación una vez finalizada, o puede agrupar prestaciones para liberarlas de una sola vez.
- Espiral (o prototipado):
 - Implica la creación de incrementos experimentales, algunos de los cuales pueden ser reelaborados en profundidad o incluso abandonados en trabajos de desarrollo posteriores.

© JMA 2020. All rights reserved

RUP



© JMA 2020. All rights reserved

Clasificación de Pruebas

- Las actividades de las pruebas pueden centrarse en comprobar el sistema en base a un objetivo o motivo específico:
 - Una función a realizar por el software.
 - Una característica no funcional como el rendimiento o la fiabilidad.
 - La estructura o arquitectura del sistema o el software.
 - Los cambios para confirmar que se han solucionado los defectos o localizar los no intencionados.
- Las pruebas se pueden clasificar como:
 - Pruebas funcionales
 - Pruebas no funcionales
 - Pruebas estructurales
 - Pruebas de mantenimiento

© JMA 2020. All rights reserved

Niveles de pruebas

- **Pruebas Unitarias o de Componentes:** verifican la funcionalidad y estructura de cada componente individualmente, una vez que ha sido codificado.
- **Pruebas de Integración:** verifican el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente, con el fin de comprobar que interactúan correctamente a través de sus interfaces, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.
- **Pruebas de Regresión:** verifican que los cambios sobre un componente de un sistema de información no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.
- **Pruebas del Sistema:** ejercitan profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- **Pruebas de Aceptación:** validan que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema, que determine su aceptación desde el punto de vista de su funcionalidad y rendimiento.

© JMA 2020. All rights reserved

Pruebas Unitarias

- Las pruebas unitarias tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente, una vez que ha sido codificado.
- Con las pruebas unitarias verificas el diseño de los programas, vigilando que no se producen errores y que el resultado de los programas es el esperado.
- Estas pruebas las efectúa normalmente la misma persona que codifica o modifica el componente y que, también normalmente, genera un juego de ensayo para probar y depurar las condiciones de prueba.
- Las pruebas unitarias constituyen la prueba inicial de un sistema y las demás pruebas deben apoyarse sobre ellas.

© JMA 2020. All rights reserved

Pruebas Unitarias

- Existen dos **enfoques** principales para el diseño de casos de prueba:
 - **Enfoque estructural o de caja blanca.** Se verifica la estructura interna del componente con independencia de la funcionalidad establecida para el mismo.
Por tanto, no se comprueba la corrección de los resultados, sólo si éstos se producen. Ejemplos de este tipo de pruebas pueden ser ejecutar todas las instrucciones del programa, localizar código no usado, comprobar los caminos lógicos del programa, etc.
 - **Enfoque funcional o de caja negra.** Se comprueba el correcto funcionamiento de los componentes del sistema de información, analizando las entradas y salidas y verificando que el resultado es el esperado. Se consideran exclusivamente las entradas y salidas del sistema sin preocuparse por la estructura interna del mismo.
- El enfoque que suele adoptarse para una prueba unitaria está claramente orientado al diseño de casos de caja blanca, aunque se complementa con caja negra.

© JMA 2020. All rights reserved

Pruebas Unitarias

- Los **pasos necesarios** para llevar a cabo las pruebas unitarias son los siguientes:
 - **Ejecutar todos los casos de prueba** asociados a cada verificación establecida en el plan de pruebas, registrando su resultado. Los casos de prueba deben contemplar tanto las condiciones válidas y esperadas como las inválidas e inesperadas.
 - **Corregir los errores o defectos encontrados y repetir las pruebas que los detectaron.** Si se considera necesario, debido a su implicación o importancia, se repetirán otros casos de prueba ya realizados con anterioridad.

© JMA 2020. All rights reserved

Pruebas Unitarias

- La prueba unitaria se da por finalizada cuando se hayan realizado todas las verificaciones establecidas y no se encuentre ningún defecto, o bien se determine su suspensión.
- Al finalizar las pruebas, obtienes las **métricas de calidad del componente** y las contrastas con las existentes antes de la modificación:
 - Número ciclomático.
 - Cobertura de código.
 - Porcentaje de comentarios.
 - Defectos hallados contra especificaciones o estándares.
 - Rendimientos.

© JMA 2020. All rights reserved

Pruebas de Integración

- Las pruebas de integración te permiten verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente, con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.
- Se trata de probar los caminos lógicos del flujo de los datos y mensajes a través de un conjunto de componentes relacionados que definen una cierta funcionalidad.
- En las pruebas de integración examinas las interfaces entre grupos de componentes o subsistemas para asegurar que son llamados cuando es necesario y que los datos o mensajes que se transmiten son los requeridos.
- Debido a que en las pruebas unitarias es necesario crear módulos auxiliares que simulen las acciones de los componentes invocados por el que se está probando, y a que se han de crear componentes "conductores" para establecer las precondiciones necesarias, llamar al componente objeto de la prueba y examinar los resultados de la prueba, a menudo se combinan los tipos de prueba unitarias y de integración.

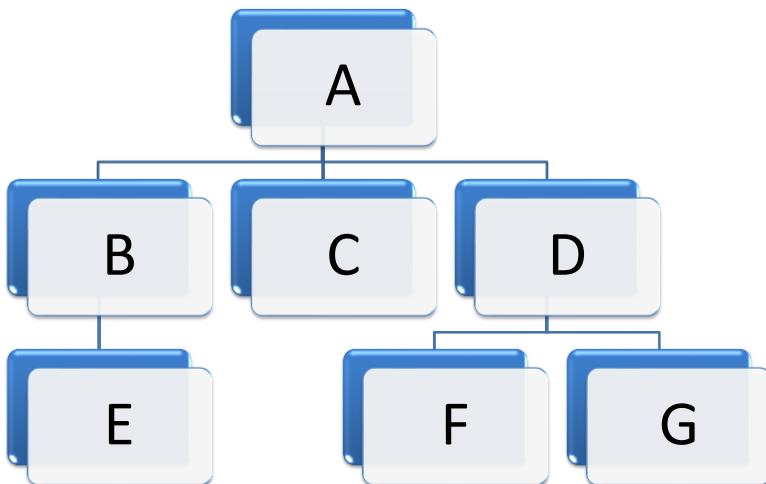
© JMA 2020. All rights reserved

Pruebas de Integración

- Los **tipos fundamentales de integración** son los siguientes:
 - **Integración incremental:** combinas el siguiente componente que debes probar con el conjunto de componentes que ya están probados y vas incrementando progresivamente el número de componentes a probar.
 - **Integración no incremental:** pruebas cada componente por separado y, luego, los integras todos de una vez realizando las pruebas pertinentes. Este tipo de integración se denomina también Big-Bang (gran explosión).
- Con el tipo de prueba incremental lo más probable es que los problemas te surjan al incorporar un nuevo componente o un grupo de componentes al previamente probado. Los problemas serán debidos a este último o a las interfaces entre éste y los otros componentes.

© JMA 2020. All rights reserved

Pruebas de Integración



© JMA 2020. All rights reserved

Estrategias de integración

- **De arriba a abajo (top-down):** el primer componente que se desarrolla y prueba es el primero de la jerarquía (A).
 - Los componentes de nivel más bajo se sustituyen por componentes auxiliares o resguardos, para simular a los componentes invocados. En este caso no son necesarios componentes conductores.
 - Una de las ventajas de aplicar esta estrategia es que las interfaces entre los distintos componentes se prueban en una fase temprana y con frecuencia.
- **De abajo a arriba (bottom-up):** en este caso se crean primero los componentes de más bajo nivel (E, F, G) y se crean componentes conductores para simular a los componentes que los llaman.
 - A continuación se desarrollan los componentes de más alto nivel (B, C, D) y se prueban. Por último dichos componentes se combinan con el que los llama (A). Los componentes auxiliares son necesarios en raras ocasiones.
 - Este tipo de enfoque permite un desarrollo más en paralelo que el enfoque de arriba a abajo, pero presenta mayores dificultades a la hora de planificar y de gestionar.
- **Estrategias combinadas:** A menudo es útil aplicar las estrategias anteriores conjuntamente. De este modo, se desarrollan partes del sistema con un enfoque "top-down", mientras que los componentes más críticos en el nivel más bajo se desarrollan siguiendo un enfoque "bottom-up".
 - En este caso es necesaria una planificación cuidadosa y coordinada de modo que los componentes individuales se "encuentren" en el centro.

© JMA 2020. All rights reserved

Pruebas de Regresión

- El objetivo de las pruebas de regresión es eliminar el efecto onda, es decir, comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.
- Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora.
- No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre otros componentes.
- Normalmente, este tipo de pruebas implica la repetición de las pruebas que ya se han realizado previamente, con el fin de asegurar que no se introducen errores que puedan comprometer el funcionamiento de otros componentes que no han sido modificados y confirmar que el sistema funciona correctamente una vez realizados los cambios.

© JMA 2020. All rights reserved

Pruebas de Regresión

- Las pruebas de regresión **pueden incluir**:
 - La repetición de los casos de pruebas que se han realizado anteriormente y están directamente relacionados con la parte del sistema modificada.
 - La revisión de los procedimientos manuales preparados antes del cambio, para asegurar que permanecen correctamente.
 - La obtención impresa del diccionario de datos de forma que se compruebe que los elementos de datos que han sufrido algún cambio son correctos.
- El **responsable** de realizar las pruebas de regresión será el equipo de desarrollo junto al técnico de mantenimiento, quién a su vez, será responsable de especificar el plan de pruebas de regresión y de evaluar los resultados de dichas pruebas.

© JMA 2020. All rights reserved

Pruebas del Sistema

- Las pruebas del sistema tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- Son pruebas de integración del sistema de información completo, y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción.
- Una vez que se han probado los componentes individuales y se han integrado, se prueba el sistema de forma global. En esta etapa pueden distinguirse diferentes tipos de pruebas, cada uno con un objetivo claramente diferenciado.

© JMA 2020. All rights reserved

Pruebas del Sistema

- **Pruebas funcionales:** dirigidas a asegurar que el sistema de información realiza correctamente todas las funciones que se han detallado en las especificaciones dadas por el usuario del sistema.
- **Pruebas de humo:** son un conjunto de pruebas aplicadas a cada nueva versión, su objetivo es validar que las funcionalidades básicas de la versión se cumplen según lo especificado. Impiden la ejecución el plan de pruebas si detectan grandes inestabilidades o si elementos clave faltan o son defectuosos.
- **Pruebas de comunicaciones:** determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Asimismo, se han de probar las interfaces hombre-máquina.
- **Pruebas de rendimiento:** consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- **Pruebas de volumen:** consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.
- **Pruebas de sobrecarga o estrés:** consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.

© JMA 2020. All rights reserved

Pruebas del Sistema

- **Pruebas de disponibilidad de datos:** consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.
- **Pruebas de usabilidad:** consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados.
- **Pruebas extremo a extremo (e2e):** consisten en interactuar con la aplicación como un usuario regular lo haría, cliente-servidor, y evaluando las respuestas para el comportamiento esperado.
- **Pruebas de configuración:** consisten en comprobar todos y cada uno de los dispositivos, en sus propiedades mínimo y máximo posibles.
- **Pruebas de operación:** consisten en comprobar la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y re-arranque del sistema, etc.
- **Pruebas de entorno:** consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- **Pruebas de seguridad:** consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.

© JMA 2020. All rights reserved

Pruebas de rendimiento



© JMA 2020. All rights reserved

Pruebas de carga, estrés y picos

- Pruebas de carga (load test): pruebas para determinar y validar la respuesta de la aplicación cuando es sometida a una carga de usuarios y/o transacciones que se espera en el ambiente de producción. Ejemplo: verificar la correcta respuesta de la aplicación ante el alta de 100 usuarios en forma simultánea. Se compara con el volumen esperado.
- Pruebas de estrés (stress test): pruebas para encontrar el volumen de datos o de tiempo en que la aplicación comienza a fallar o es incapaz de responder a las peticiones. Son pruebas de carga o rendimiento, pero superando los límites esperados en el ambiente de producción y/o determinados en las pruebas. Ejemplo: encontrar la cantidad de usuarios simultáneos, en que la aplicación deja de responder (cuelgue o time out) en forma correcta a todas las peticiones.
- Pruebas de picos (spike testing): es un sub tipo de prueba de estrés que mide el rendimiento del software bajo un «pico» significativo y repentino o una carga de trabajo creciente como la de los usuarios simulados. Indica si el software puede manejar ese aumento abrupto de la carga de trabajo de forma repetida y rápida.

© JMA 2020. All rights reserved

Pruebas de resistencia, volumen y escalabilidad

- Pruebas de resistencia (soak testing, endurance testing): evalúa el rendimiento del software durante un periodo prolongado bajo una carga de trabajo regular y fija, determina cuánto tiempo puede soportar el software una carga de trabajo constante para proporcionar sostenibilidad a largo plazo. Durante estas pruebas, los equipos de pruebas supervisan los KPI como las fugas de memoria, de proceso, etc. Las pruebas de resistencia también analizan los tiempos de respuesta y el rendimiento tras un uso prolongado para mostrar si estas métricas son consistentes.
- Pruebas de volumen (volume testing): comprueban la eficacia del software cuando se somete a grandes volúmenes de datos. Comprueba la pérdida de datos, el tiempo de respuesta del sistema, la fiabilidad del almacenamiento de datos, etc.
- Pruebas de escalabilidad (scalability testing): miden la eficacia del software a la hora de manejar una cantidad creciente de carga de trabajo, añadiendo volumen de datos o usuarios de forma gradual mientras se supervisa el rendimiento del software. La prueba informará sobre el comportamiento cuando aumenten o disminuyan los atributos de rendimiento del software.

© JMA 2020. All rights reserved

Pruebas de Aceptación

- El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema, que determine su aceptación desde el punto de vista de su funcionalidad y rendimiento.
- Las pruebas de aceptación son preparadas por el usuario del sistema y el equipo de desarrollo, aunque la ejecución y aprobación final corresponde al usuario.
- Estas pruebas van dirigidas a comprobar que el sistema cumple los requisitos de funcionamiento esperado recogidos en el catálogo de requisitos y en los criterios de aceptación del sistema de información, y conseguir la aceptación final del sistema por parte del usuario.

© JMA 2020. All rights reserved

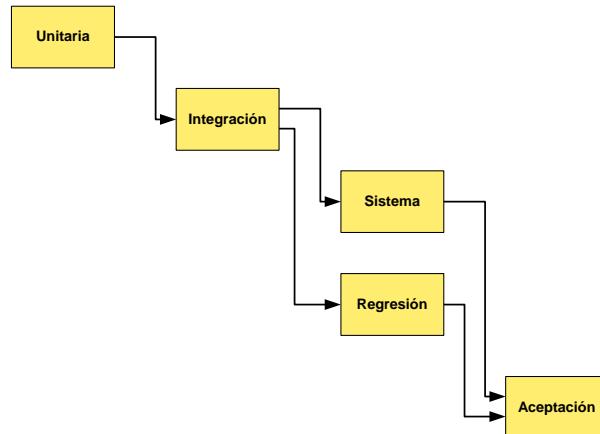
Pruebas de Aceptación

- Previamente a la realización de las pruebas, el responsable de usuarios revisa los criterios de aceptación que se especificaron previamente en el plan de pruebas del sistema y dirige las pruebas de aceptación final.
- La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas, el cual define las verificaciones a realizar y los casos de prueba asociados.
- Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta, a su vez, los requisitos no funcionales relacionados con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.
- La formalidad de estas pruebas dependerá en mayor o menor medida de cada organización, y vendrá dada por la criticidad del sistema, el número de usuarios implicados en las mismas y el tiempo del que se disponga para llevarlas cabo, entre otros.

© JMA 2020. All rights reserved

Niveles de pruebas y orden de ejecución.

- De tal forma que la secuencia de pruebas es:



© JMA 2020. All rights reserved

Prueba exploratoria

- Incluso los esfuerzos de automatización de pruebas más diligentes no son perfectos. A veces se pierden ciertos casos extremos en sus pruebas automatizadas. A veces es casi imposible detectar un error en particular escribiendo una prueba unitaria. Ciertos problemas de calidad ni siquiera se hacen evidentes en las pruebas automatizadas (como en el diseño o la usabilidad).
- Las pruebas exploratorias es un enfoque de prueba manual que enfatiza la libertad y creatividad del probador para detectar problemas de calidad en un sistema en ejecución.
 - Simplemente tomate un tiempo en un horario regular, arremágate e intenta romper la aplicación.
 - Usa una mentalidad destructiva y encuentra formas de provocar problemas y errores en la aplicación.
 - Ten en cuenta los errores, los problemas de diseño, los tiempos de respuesta lentos, los mensajes de error faltantes o engañosos y, en general, todo lo que pueda molestarte como usuario de una aplicación.
 - Documenta todo lo que encuentre para más adelante.
- La buena noticia es que se puede automatizar la mayoría de los hallazgos con pruebas automatizadas. Escribir pruebas automatizadas para los errores que se detectan asegura que no habrá regresiones a ese error en el futuro. Además, ayuda a reducir la causa raíz de ese problema durante la corrección de errores.

© JMA 2020. All rights reserved

Pruebas de mutaciones

- Los pruebas de mutaciones son las pruebas de las pruebas unitarias y el objetivo es tener una idea de la calidad de las pruebas en cuanto a fiabilidad.
- Su funcionamiento es relativamente sencillo: la herramienta que se utilice debe generar pequeños cambios en el código fuente. A estos pequeños cambios se les conoce como mutaciones y crean mutantes.
- Una vez creados los mutantes, se lanzan todos los tests:
 - Si los test unitarios fallan, es que han sido capaces de detectar ese cambio de código. En este caso el mutante se considera eliminado.
 - Si, por el contrario, los test unitarios pasan, el mutante sobrevive y la fiabilidad (y calidad) de los tests unitarios queda en entredicho.
- Los test de mutaciones presentan informes del porcentaje de mutantes detectados: cuanto más se acerque este porcentaje al 100%, mayor será la calidad de nuestros test unitarios.

© JMA 2020. All rights reserved

Pirámide de pruebas



© JMA 2020. All rights reserved

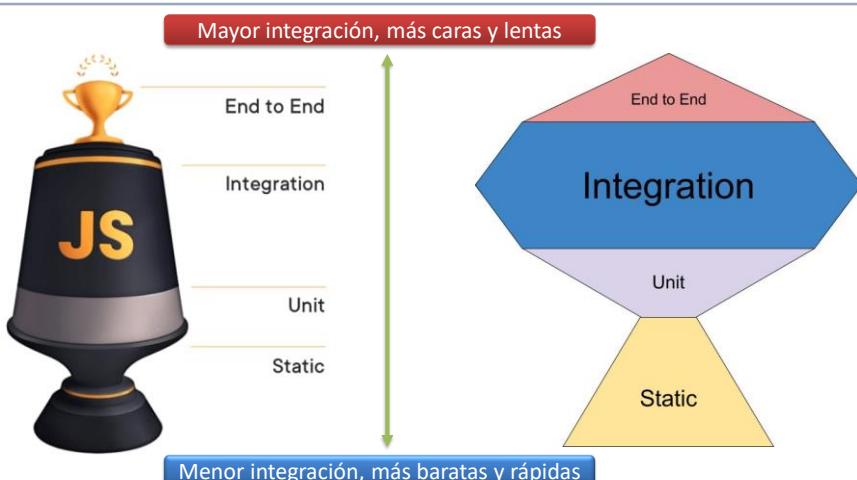
<https://martinfowler.com/bliki/TestPyramid.html>

El Trofeo de Pruebas

- Testing Trophy es un método de pruebas propuesto por Kent C. Dodds para aplicaciones web. Se trata de escribir suficientes pruebas, no muchas, pero si las pruebas correctas: proporciona la mejor combinación de velocidad, costo y confiabilidad.
- Se superpondrán las siguientes técnicas:
 - Usar un sistema de captura de errores de tipo, estilo y de formato utilizando linters, formateadores de errores y verificadores de tipo (ESLint, SonarQube, ...).
 - Escribir pruebas unitarias efectivas que apunten solo al comportamiento crítico y la funcionalidad de la aplicación.
 - Desarrollar pruebas de integración para auditar la aplicación de manera integral y asegurarse de que todo funcione correctamente en armonía.
 - Crear pruebas funcionales de extremo a extremo (e2e) para pruebas de interacción automatizadas de las rutas críticas y los flujos de trabajo más utilizados por los usuarios.

© JMA 2020. All rights reserved

Testing Trophy



© JMA 2020. All rights reserved

Aprender con pruebas unitarias

- La incorporación de código de tercero es complicado, hay que aprenderlo primero e integrarlo después. Hacer las dos cosa a la vez es el doble de complicado.
- Utilizar pruebas unitarias en el proceso de aprendizaje (*pruebas de aprendizaje* según Jim Newkirk) aporta importantes ventajas:
 - La inmediatez de las pruebas unitarias y sus entornos.
 - Realizar “pruebas de concepto” para comprobar si el comportamiento se corresponde con lo que hemos entendido, permitiéndonos clarificarlo.
 - Experimentar para encontrar los mejores escenarios de integración.
 - Permite saber si un fallo es nuestro, de la librería o del uso inadecuado de la librería.

© JMA 2020. All rights reserved

Pruebas de aprendizaje

- Las pruebas de aprendizaje no suponen un coste adicional, es parte del coste de aprendizaje que, en todo caso, lo minoran.
- Es mas, las pruebas de aprendizaje son rentables. Ante la aparición de nuevas versiones del código ajeno, ejecutar la batería de pruebas de aprendizaje valida el impacto de la adopción de la nueva versión: detecta cambios relevantes, efectos negativos en las integraciones, ...
- Las pruebas de aprendizaje no sustituyen al conjunto de pruebas que respaldan los límites establecidos.

© JMA 2020. All rights reserved

Análisis estático con herramientas

- El objetivo del análisis estático es detectar defectos en el código fuente y en los modelos de software.
- El análisis estático se realiza sin que la herramienta llegue a ejecutar el software objeto de la revisión, como ocurre en las pruebas dinámicas, centrándose más en como está escrito el código que en como se ejecuta el código.
- El análisis estático permite identificar defectos difíciles de encontrar mediante pruebas dinámicas.
- Al igual que con las revisiones, el análisis estático encuentra defectos en lugar de fallos.
- Las herramientas de análisis estático analizan el código del programa (por ejemplo, el flujo de control y flujo de datos) y las salidas generadas (tales como HTML o XML).
- Algunos de los posibles aspectos que pueden ser comprobados con análisis estático:
 - Reglas, estándares de programación y buenas prácticas.
 - Diseño de un programa (análisis de flujo de control).
 - Uso de datos (análisis del flujo de datos).
 - Complejidad de la estructura de un programa (métricas, por ejemplo valor ciclomático).

© JMA 2020. All rights reserved

Valor del análisis estático

- La detección temprana de defectos antes de la ejecución de las pruebas.
- Advertencia temprana sobre aspectos sospechosos del código o del diseño mediante el cálculo de métricas, tales como una medición de alta complejidad.
- Identificación de defectos que no se encuentran fácilmente mediante pruebas dinámicas.
- Detectar dependencias e inconsistencias en modelos de software, como enlaces.
- Mantenibilidad mejorada del código y del diseño.
- Prevención de defectos, si se aprende la lección en la fase de desarrollo.

© JMA 2020. All rights reserved

Defectos habitualmente detectados

- Referenciar una variable con un valor indefinido.
- Interfaces inconsistentes entre módulos y componentes.
- Variables que no se utilizan o que se declaran de forma incorrecta.
- Código inaccesible (muerto).
- Ausencia de lógica o lógica errónea (posibles bucles infinitos).
- Construcciones demasiado complicadas.
- Infracciones de los estándares de programación.
- Vulnerabilidad de seguridad.
- Infracciones de sintaxis del código y modelos de software.

© JMA 2020. All rights reserved

Ejecución del análisis estático

- Las herramientas de análisis estático generalmente las utilizan los desarrolladores (cotejar con las reglas predefinidas o estándares de programación) antes y durante las pruebas unitarias y de integración, o durante la comprobación del código.
- Las herramientas de análisis estático pueden producir un gran número de mensajes de advertencias que deben ser bien gestionados para permitir el uso más efectivo de la herramienta.
- Los compiladores pueden constituir un soporte para los análisis estáticos, incluyendo el cálculo de métricas.
- El Compilador detecta errores sintácticos en el código fuente de un programa, crea datos de referencia del programa (por ejemplo lista de referencia cruzada, llamada jerárquica, tabla de símbolos), comprueba la consistencia entre los tipos de variables y detecta variables no declaradas y código inaccesible (código muerto).
- El Analizador trata aspectos adicionales tales como: Convenciones y estándares, Métricas de complejidad y Acoplamiento de objetos.

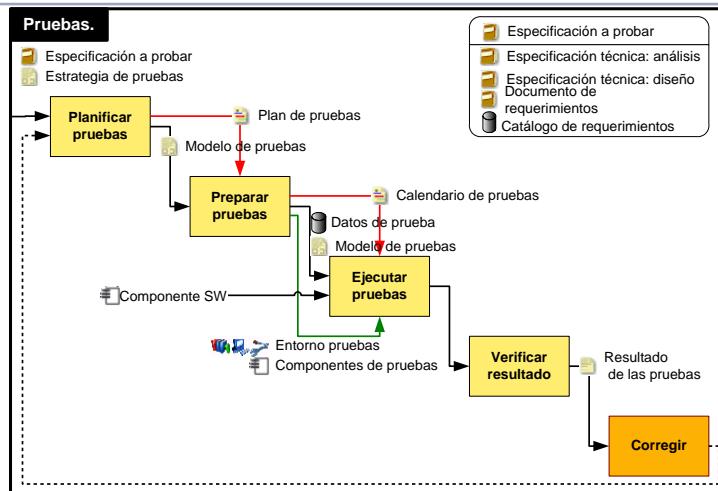
© JMA 2020. All rights reserved

Principios F.I.R.S.T.

- El principio FIRST fue definido por Robert Cecil Martin en su libro Clean Code. Este autor, entre otras muchas cosas, es conocido por ser uno de los escritores del Agile Manifesto, escrito hace más de 15 años y que a día de hoy se sigue teniendo muy en cuenta a la hora de desarrollar software.
 - Fast: Los tests deben ser rápidos, del orden de milisegundos, hay cientos de tests en un proyecto.
 - Isolated/Independent (Aislado/Independiente). Los tests no deben depender del entorno ni de ejecuciones de tests anteriores.
 - Repeatable. Los tests deben ser repetibles y ante la misma entrada de datos, los mismos resultados.
 - Self-Validating. Los tests tienen que ser autovalidados, es decir, NO debe de existir la intervención humana en la validación
 - Thorough and Timely (Completo y oportuno). Los tests deben de cubrir el escenario propuesto, no el 100% del código, y se han de realizar en el momento oportuno

© JMA 2020. All rights reserved

Metodología



© JMA 2020. All rights reserved

Diseñar la prueba

- Para diseñar la prueba empiezas por identificar y describir los casos de prueba de cada componente.
- La selección de las técnicas de pruebas depende factores adicionales como pueden ser requisitos contractuales o normativos, documentación disponible, tiempo, presupuesto, conocimientos, experiencia, ...
- Cuando dispongas de los casos de prueba, identificas y estructuras los procedimientos de prueba describiendo cómo ejecutar los casos de prueba.

© JMA 2020. All rights reserved

Patrones

- Los casos de prueba se pueden estructurar siguiendo diferentes patrones:
 - ARRANGE-ACT-ASSERT: Preparar, Actuar, Afirmar
 - GIVEN-WHEN-THEN: Dado, Cuando, Entonces
 - BUILD-OPERATE-CHECK: Generar, Operar, Comprobar
- Aunque con diferencias conceptuales, todos dividen el proceso en tres fases:
 - Una fase inicial donde montar el escenario de pruebas que hace que el resultado sea predecible.
 - Una fase intermedia donde se realizan las acciones que son el objetivo de la prueba.
 - Una fase final donde se comparan los resultados con el escenario previsto. Pueden tomar la forma de:
 - Aserción: Es una afirmación sobre el resultado que puede ser cierta o no.
 - Expectativa: Es la expresión del resultado esperado que puede cumplirse o no.

© JMA 2020. All rights reserved

Preparación mínima

- La sección de preparación, con la entrada del caso de prueba, debe ser lo más sencilla posible, lo imprescindible para comprobar el comportamiento que se está probando.
- Las pruebas se hacen más resistentes a los cambios futuros en el código base y más cercano al comportamiento de prueba que a la implementación.
- Las pruebas que incluyen más información de la necesaria tienen una mayor posibilidad de incorporar errores en la prueba y pueden hacer confusa su intención. Al escribir pruebas, el usuario quiere centrarse en el comportamiento. El establecimiento de propiedades adicionales en los modelos o el empleo de valores distintos de cero cuando no es necesario solo resta de lo que se quiere probar.

© JMA 2020. All rights reserved

Actuación mínima

- Al escribir las pruebas hay que evitar introducir condiciones lógicas como if, switch, while, for, etc.
- Minimiza la posibilidad de incorporar un error a las pruebas.
- El foco está en el resultado final, en lugar de en los detalles de implementación.
- Al incorporar lógica al conjunto de pruebas, aumenta considerablemente la posibilidad de agregar un error. Cuando se produce un error en una prueba, se quiere saber realmente que algo va mal con el código probado y no en el código que prueba. En caso contrario, restan confianza y las pruebas en las que no se confía no aportan ningún valor.
- El objetivo de la prueba debe ser único, si la lógica en la prueba parece inevitable, denota que el objetivo es múltiple y hay que considerar la posibilidad de dividirla en dos o más pruebas diferentes.

© JMA 2020. All rights reserved

Comprobación mínima

- Al escribir las pruebas, hay que intentar comprobar una única cosa, es decir, incluir solo una aserción por prueba.
 - Si se produce un error en una aserción, no se evalúan las aserciones posteriores.
 - Garantiza que no se estén declarando varios casos en las pruebas.
 - Proporciona la imagen exacta de por qué se producen errores en las pruebas.
- Al incorporar varias aserciones en un caso de prueba, no se garantiza que se ejecuten todas. Es un todas o ninguna, se sabe por cual fallo pero no si el resto también falla o es correcto, proporcionando la imagen parcial.
- Una excepción común a esta regla es cuando la validación cubre varios aspectos. En este caso, suele ser aceptable que haya varias aserciones para asegurarse de que el resultado está en el estado que se espera que esté.
- Los enfoques comunes para usar solo una aserción incluyen:
 - Crear una prueba independiente para cada aserción.
 - Usar pruebas con parámetros.

© JMA 2020. All rights reserved

Diseñar para probar

- Patrones:
 - Doble herencia
 - Inversión de Control
 - Inyección de Dependencias
 - Modelo Vista Controlador (MVC)
 - Model View ViewModel (MVVM)
- Metodologías:
 - Desarrollo Guiado por Pruebas (TDD)
 - Desarrollo Dirigido por Comportamiento (BDD)

© JMA 2020. All rights reserved

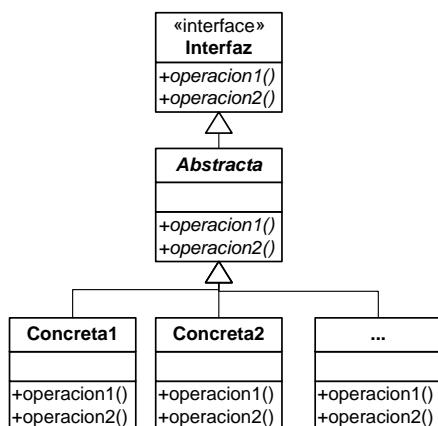
Programar con interfaces

- La herencia de clase define la implementación de una clase a partir de otra (excepto métodos abstractos). La implementación de interfaz define como se llamará el método o propiedad, pudiendo escribir distinto código en clases no relacionadas.
- Reutilizar la implementación de la clase base es la mitad de la historia.
- Programar para las interfaz, no para la herencia. Favorecer la composición antes que la herencia.
- Ventajas:
 - Reducción de dependencias.
 - El cliente desconoce la implementación.
 - La vinculación se realiza en tiempo de ejecución.
 - Da consistencia (contrato).
- Desventaja:
 - Indirecciónamiento.

© JMA 2020. All rights reserved

Doble Herencia

- Problema:
 - Mantener las clases que implementan como internas del proyecto (internal o Friend), pero la interfaz pública.
 - Organizar clases que tienen un comportamiento parecido para que sea consistente.
- Solución:
 - Clase base es abstracta.
 - La clase base puede heredar de más de una interfaz.
 - Una vez que están escritos los métodos, verifico si hay duplicación en las clases hijas.



© JMA 2020. All rights reserved

Inversión de Control

- Inversión de control (Inversion of Control en inglés, IoC) es un concepto junto con unas técnicas de programación:
 - en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales,
 - en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones.
- Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones.
- En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir. Técnicas de implementación:
 - Service Locator: es un componente (contenedor) que contiene referencias a los servicios y encapsula la lógica que los localiza dichos servicios.
 - Inyección de dependencias.

© JMA 2020. All rights reserved

Inyección de Dependencias

- Las dependencias son expresadas en términos de interfaces en lugar de clases concretas y se resuelven dinámicamente en tiempo de ejecución.
- La Inyección de Dependencias (en inglés Dependency Injection, DI) es un patrón de arquitectura orientado a objetos, en el que se inyectan objetos a una clase en lugar de ser la propia clase quien cree el objeto, básicamente recomienda que las dependencias de una clase no sean creadas desde el propio objeto, sino que sean configuradas desde fuera de la clase.
- La inyección de dependencias (DI) procede del patrón de diseño más general que es la Inversión de Control (IoC).
- Al aplicar este patrón se consigue que las clases sean independientes unas de otras e incrementando la reutilización y la extensibilidad de la aplicación, además de facilitar las pruebas unitarias de las mismas.
- Desde el punto de vista de Java o .NET, un diseño basado en DI puede implementarse mediante el lenguaje estándar, dado que una clase puede leer las dependencias de otra clase por medio del Reflection y crear una instancia de dicha clase inyectándole sus dependencias.

© JMA 2020. All rights reserved

Simulación de objetos

- Las dependencias con sistemas externos afectan a la complejidad de la estrategia de pruebas, ya que es necesario contar con sustitutos de estos servicios externos durante el desarrollo. Ejemplos típicos de estas dependencias son Servicios Web, Sistemas de envío de correo, Fuentes de Datos o simplemente dispositivos hardware.
- Estos sustitutos, muchas veces son exactamente iguales que el servicio original, pero en otro entorno o son simuladores que exponen el mismo interfaz pero realmente no realizan las mismas tareas que el sistema real, o las realizan contra un entorno controlado.
- Para poder emplear la técnica de simulación de objetos se debe diseñar el código a probar de forma que sea posible trabajar con los objetos reales o con los objetos simulados:
 - Doble herencia
 - IoC: Inversión de Control (Inversion Of Control)
 - DI: Inyección de Dependencias (Dependency Injection)
 - Objetos Mock

© JMA 2020. All rights reserved

Dobles de prueba

- La regla de oro de las pruebas unitarias, es que una unidad (unit) tiene que ser testeada sin utilizar ninguna de sus dependencias.
- Siguiendo la misma regla de oro, las pruebas de integración y sistema deben estar aisladas de sus dependencias salvo cuando se estén probando dichas dependencias. Así mismo, el resultado de las pruebas debe ser previsible.
- Entre las ventajas de esta aproximación se encuentran:
 - Devuelven resultados determinísticos
 - Permiten crear o reproducir determinados estados (por ejemplo errores de conexión)
 - Obtienen resultados mucho mas rápidamente y a menor coste, incluso offline.
 - Permiten el inicio temprano de las pruebas incluso cuando las dependencias todavía no están disponibles.
 - Permiten incluir atributos o métodos exclusivamente para el testeo.

© JMA 2020. All rights reserved

Dobles de prueba

- La forma de establecer los valores esperados y “memorizar” el valor con el que se ha llamado al simulador para posteriormente verificarlo se ha generalizado dando lugar a un marco de trabajo que permite definir objetos simulados sin necesidad de crear explícitamente el código que verifica cada uno de los valores.
- Los dobles de prueba son objetos que siempre realizan las mismas tareas:
 - Implementan un interfaz dado
 - Permiten establecer los valores esperados (tanto de entrada como de salida)
 - Permiten establecer el comportamiento (para lanzar excepciones en casos concretos)
 - Memorizan los valores con los que se llama a cada uno de sus miembros
 - Permiten verificar si los valores esperados coinciden con los recibidos

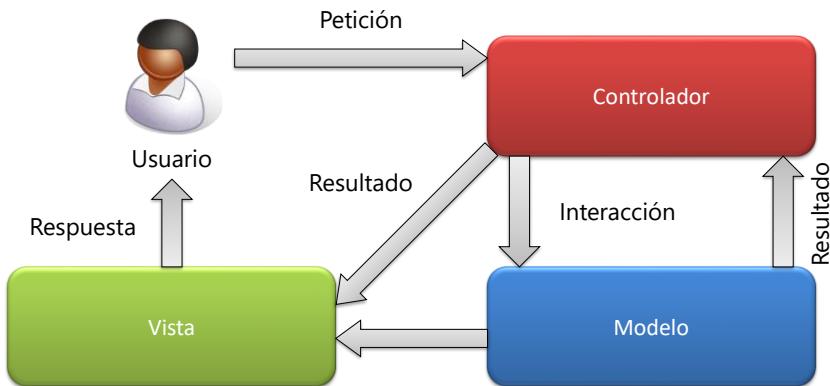
© JMA 2020. All rights reserved

Dobles de prueba

- **Fixture:** Es el término se utiliza para hablar de los datos de contexto de las pruebas, aquellos que se necesitan para construir el escenario que requiere la prueba.
- **Dummy:** Objeto que se pasa como argumento pero nunca se usa realmente. Normalmente, los objetos dummy se usan sólo para llenar listas de parámetros.
- **Fake:** Objeto que tiene una implementación que realmente funciona pero, por lo general, usa una simplificación que le hace inapropiado para producción (como una base de datos en memoria por ejemplo).
- **Stub:** Objeto que proporciona respuestas predefinidas a llamadas hechas durante los tests, frecuentemente, sin responder en absoluto a cualquier otra cosa fuera de aquello para lo que ha sido programado. Los stubs pueden también grabar información sobre las llamadas (**spy**).
- **Mock:** Objeto pre programado con expectativas que conforman la especificación de cómo se espera que se reciban las llamadas. Son más complejos que los stubs aunque sus diferencias son sutiles.

© JMA 2020. All rights reserved

El patrón MVC



© JMA 2020. All rights reserved

El patrón MVC



- Representación de los **datos del dominio**
- Lógica de **negocio**
- Mecanismos de **persistencia**



- **Interfaz** de usuario
- Incluye elementos de **interacción**



- **Intermediario** entre Modelo y Vista
- **Mapea acciones** de usuario → acciones del Modelo
- **Selecciona** las vistas y les **suministra** información

© JMA 2020. All rights reserved

Desarrollo Guiado por Pruebas (TDD)

- El Desarrollo Guiado por Pruebas, es una técnica de programación (definida por KentBeck); consistente en desarrollar primero el código que pruebe una característica o funcionalidad deseada antes que el código que implementa dicha funcionalidad.
- El objetivo a lograr es que no exista ninguna funcionalidad que no esté avalada por una prueba.
- Lo primero que hay que aprender de TDD son sus reglas básicas:
 - No añadir código sin escribir antes una prueba que falle
 - Eliminar el Código Duplicado empleando Refactorización

© JMA 2020. All rights reserved

Ritmo TDD

- TDD invita a seguir una serie de tareas ordenadas, que a menudo se denomina ritmo TDD, y que se basa en los siguientes pasos:
 1. Escribir una prueba que demuestre la necesidad de escribir código.
 2. Escribir el mínimo código para que el código de pruebas compile
 3. Implementar exclusivamente la funcionalidad demandada por las pruebas
 4. Mejorar el código (Refactoring) sin añadir funcionalidad
 5. Volver al primer paso
- Este ritmo permite formalizar las tareas que se han de realizar para conseguir un código fácil de mantener, bien diseñado y que se puede probar automáticamente.

© JMA 2020. All rights reserved

Beneficios de TDD

- Reducen el número de errores y bugs ya que éstos, aplicando TDD, se detectan antes incluso de crearlos.
- Facilitan entender el código y que, eligiendo una buena nomenclatura, sirven de documentación.
- Facilitan mantener el código:
 - Protege ante cambios, los errores que surgen al aplicar un cambio se detectan (y corrigen) antes de subir ese cambio.
 - Protegen ante errores de regresión (rollbacks a versiones anteriores).
 - Dan confianza.
- Facilitan desarrollar ciñéndose a los requisitos.
- Ayudan a encontrar inconsistencias en los requisitos
- Ayudan a especificar comportamientos
- Ayudan a refactorizar para mejorar la calidad del código (Clean code)
- A medio/largo plazo aumenta (y mucho) la productividad.

© JMA 2020. All rights reserved

Estrategia RED – GREEN

- Se recomienda una estrategia de test unitarios conocida como **RED** (fallo) – **GREEN** (éxito), es especialmente útil en equipos de desarrollo ágil.
- Una vez que entendamos la lógica y la intención de un test unitario, hay que seguir estos pasos:
 - Escribe el código del test (**Stub**) para que compile (pase de **RED** a **GREEN**)
 - Inicialmente la compilación fallará **RED** debido a que falta código
 - Implementa sólo el código necesario para que compile **GREEN** (aún no hay implementación real).
 - Escribe el código del test para que se **ejecute** (pase de **RED** a **GREEN**)
 - Inicialmente el test fallará **RED** ya que no existe funcionalidad.
 - Implementa la funcionalidad que va a probar el test hasta que se ejecute adecuadamente **GREEN**.
 - **Refactoriza** el test y el código una vez que este todo **GREEN** y la solución vaya evolucionando.

© JMA 2020. All rights reserved

Ritmo TDD



© JMA 2020. All rights reserved

Refactorizar el código en pruebas

- Una refactorización es un cambio que está pensado para que el código se ejecute mejor o para que sea más fácil de comprender.
- No está pensado para alterar el comportamiento del código y, por tanto, no se cambian las pruebas.
- Se recomienda realizar los pasos de refactorización independientemente de los pasos que amplían la funcionalidad.
- Mantener las pruebas sin cambios aporta la confianza de no haber introducido errores accidentalmente durante la refactorización.

© JMA 2020. All rights reserved

Desarrollo Dirigido por Comportamiento (BDD)

- El Desarrollo Dirigido por Comportamiento (Behaviour Driver Development) es una evolución de TDD (Test Driven Development o Desarrollo Dirigido por Pruebas), el concepto de BDD fue inicialmente introducido por Dan North como respuesta a los problemas que surgían al enseñar TDD.
- En BDD también vamos a escribir las pruebas antes de escribir el código fuente, pero en lugar de pruebas unitarias, lo que haremos será escribir pruebas que verifiquen que el comportamiento del código es correcto desde el punto de vista de negocio. Tras escribir las pruebas escribimos el código fuente de la funcionalidad que haga que estas pruebas pasen correctamente. Después refactorizamos el código fuente.
- Partiremos de historias de usuario, siguiendo el modelo “Como [rol] quiero [característica] para [los beneficios]”. A partir de aquí, en lugar de describir en ‘lenguaje natural’ lo que tiene que hacer esa nueva funcionalidad, vamos a usar un lenguaje ubicuo (un lenguaje semiformal que es compartido tanto por desarrolladores como personal no técnico) que nos va a permitir describir todas nuestras funcionalidades de una única forma.

© JMA 2020. All rights reserved

BDD

- Para empezar a hacer BDD sólo nos hace falta conocer 5 palabras, con las que construiremos sentencias con las que vamos a describir las funcionalidades:
 - Feature (característica): Indica el nombre de la funcionalidad que vamos a probar. Debe ser un título claro y explícito. Incluimos aquí una descripción en forma de historia de usuario: “Como [rol] quiero [característica] para [los beneficios]”. Sobre esta descripción empezaremos a construir nuestros escenarios de prueba.
 - Scenario: Describe cada escenario que vamos a probar.
 - Given (dado): Provee el contexto para el escenario en que se va a ejecutar el test, tales como el punto donde se ejecuta el test, o prerequisitos en los datos. Incluye los pasos necesarios para poner al sistema en el estado que se desea probar.
 - When (cuando): Especifica el conjunto de acciones que lanzan el test. La interacción del usuario que acciona la funcionalidad que deseamos testear.
 - Then (entonces): Especifica el resultado esperado en el test. Observamos los cambios en el sistema y vemos si son los deseados.

© JMA 2020. All rights reserved

Desarrollo Dirigido por Tests de Aceptación (ATDD)

- El Desarrollo Dirigido por Test de Aceptación (ATDD), técnica conocida también como Story Test-Driven Development (STDD), es una variación del TDD pero a un nivel diferente.
- Las pruebas de aceptación o de cliente son el criterio escrito de que un sistema cumple con el funcionamiento esperado y los requisitos de negocio que el cliente demanda. Son ejemplos escritos por los dueños de producto. Es el punto de partida del desarrollo en cada iteración.
- ATDD/STDD es una forma de afrontar la implementación de una manera totalmente distinta a las metodologías tradicionales. Cambia el punto de partida, la forma de recoger y formalizar las especificaciones, sustituye los requisitos escritos en lenguaje natural (nuestro idioma) por historias de usuario con ejemplos concretos ejecutables de como el usuario utilizará el sistema, que en realidad son casos de prueba. Los ejemplos ejecutables surgen del consenso entre los distintos miembros del equipo y el usuario final.
- La lista de ejemplos (pruebas) de cada historia, se escribe en una reunión que incluye a dueños de producto, usuarios finales, desarrolladores y responsables de calidad. Todo el equipo debe entender qué es lo que hay que hacer y por qué, para concretar el modo en que se certifica que el software lo hace.

© JMA 2020. All rights reserved

ATDD

- El algoritmo o ritmo es el mismo de tres pasos que en el TDD practicado exclusivamente por desarrolladores pero a un nivel superior.
- En ATDD hay dos prácticas claves:
 - Antes de implementar (fundamental lo de antes de implementar) una necesidad, requisito, historia de usuario, etc., los miembros del equipo colaboran para crear escenarios, ejemplos, de cómo se comportará dicha necesidad.
 - Después, el equipo convierte esos escenarios en pruebas de aceptación automatizadas. Estas pruebas de aceptación típicamente se automatizan usando Selenium o similares, “frameworks” como Cucumber, etc.

© JMA 2020. All rights reserved

Data Driven Testing (DDT)

- Se basa en la creación de tests para ejecutarse en simultáneo con sus conjuntos de datos relacionados en un framework. El framework provee una lógica de test reusable para reducir el mantenimiento y mejorar la cobertura de test. La entrada y salida (del criterio de test) pueden ser resguardados en uno o más lugares del almacenamiento central o bases de datos, el formato real y la organización de los datos serán específicos para cada caso.
- Todo lo que tiene potencial de cambiar (también llamado "variabilidad," e incluye elementos como el entorno, puntos de salida, datos de test, ubicaciones, etc) está separado de la lógica del test (scripts) y movido a un 'recurso externo'. Esto puede ser configuración o conjunto de datos de test. La lógica ejecutada en el script está dictada por los valores.
- Los datos incluyen variables usadas tanto para la entrada como la verificación de la salida. En casos avanzados (y maduros) los entornos de automatización pueden ser obtenidos desde algún sistema usando los datos reales o un "sniffer", el framework DDT por lo tanto ejecuta pruebas sobre la base de lo obtenido produciendo una herramienta de test automáticos para regresión.

© JMA 2020. All rights reserved

MÉTRICAS PARA EL PROCESO DE PRUEBAS DE SOFTWARE

© JMA 2020. All rights reserved

Métricas de código

- La mayor complejidad de las aplicaciones de software moderno también aumenta la dificultad de hacer que el código confiable y fácil de mantener.
- Las métricas de código son un conjunto de medidas de software que proporcionan a los programadores una mejor visión del código que están desarrollando.
- Aprovechando las ventajas de las métricas del código, los desarrolladores pueden entender qué tipos o métodos deberían rehacerse o más pruebas.
- Los equipos de desarrollo pueden identificar los posibles riesgos, comprender el estado actual de un proyecto y realizar un seguimiento del progreso durante el desarrollo de software.
- Los desarrolladores pueden usar el IDE para generar datos de métricas de código que medir la complejidad y el mantenimiento del código administrado.

© JMA 2020. All rights reserved

Cobertura de código

- La herramienta de cobertura de código Java [EclEmma](#) (basa en la biblioteca de cobertura de código JaCoCo), integrada en Eclipse, lleva el análisis de cobertura de código directamente al entorno de trabajo:
 - Descripción general de la cobertura: La vista Cobertura enumera los resúmenes de cobertura para los proyectos Java, lo que permite profundizar en el nivel de método.
 - Resaltado de fuente: El resultado de una sesión de cobertura también es directamente visible en los editores de fuente de Java. Un código de color personalizable resalta las líneas totalmente o parcialmente cubiertas y las no cubiertas. Esto funciona tanto para código fuente propio como para fuentes adjuntas a bibliotecas externas instrumentadas.
 - Importación y exportación: los datos de cobertura se pueden exportar en formato HTML, XML o CSV o como archivos de datos de ejecución de JaCoCo (*.exec para su importación).
- Para el análisis para la cobertura de la prueba admite:
 - Diferentes contadores.
 - Varias sesiones de cobertura.
 - Fusionar sesiones.

© JMA 2020. All rights reserved

Cobertura de código

The screenshot shows the Eclipse IDE interface with several tabs open. The main editor tab contains Java code for a chess game, specifically a movement validation method. Below the code, a status bar indicates 'Finished after 0.443 seconds' and 'Runs: 40/48 Errors: 1 Failures: 0'. A 'Coverage' tab is also visible at the bottom of the editor area. To the right of the editor, there's a detailed coverage report table:

Element	Coverage	Covered Methods	Mixed Methods	Total Methods
Juego	74.1%	164	31	200
_ Juego.java	68.0%	67	31	98
_ Tablero.java	79.3%	87	25	122
_ Movimiento.java	74.0%	9	13	13
_ JuegoValidation.java	82.3%	51	11	62
_ com.example.juegos.JuegoValidation	97.9%	46	1	47
_ com.example.juegos.JuegoValidation\$1	90.0%	10	1	11
_ com.example.juegos.JuegoValidation\$2	100.0%	1	0	1
_ Afan.java	100.0%	2	0	2
_ Caballo.java	100.0%	2	0	2
_ Dama.java	100.0%	2	0	2

© JMA 2020. All rights reserved

Contadores de cobertura

- El informe utiliza un conjunto de contadores diferentes para calcular métricas de cobertura. Todos estos contadores se derivan de la información contenida en archivos de clase Java que básicamente son instrucciones de código de bytes de Java e información de depuración opcionalmente incrustada en archivos de clase.
- Este enfoque permite una instrumentación y un análisis eficientes sobre la marcha de las aplicaciones, incluso cuando no se dispone de código fuente.
- En la mayoría de los casos, la información recopilada se puede asignar al código fuente y visualizar hasta la granularidad de nivel de línea.
- De todos modos, existen limitaciones para este enfoque. Los archivos de clase deben compilarse con información de depuración para calcular la cobertura de nivel de línea y proporcionar un resultado de fuente. No todas las construcciones del lenguaje Java se pueden compilar directamente en el código de bytes correspondiente. En tales casos, el compilador de Java crea los llamados código sintético que a veces da como resultado resultados inesperados de cobertura de código.

© JMA 2020. All rights reserved

Contadores de cobertura

- Instrucciones (Cobertura C0) - Instructions counters
 - Los recuentos de unidades más pequeñas son instrucciones de código de un solo byte de Java. La cobertura de instrucciones proporciona información sobre la cantidad de código que se ha ejecutado o se ha perdido. Esta métrica es completamente independiente del formato de origen y siempre está disponible, incluso en ausencia de información de depuración en los archivos de clase.
- Bifurcaciones (Cobertura C1) - Branches counters
 - También calcula la cobertura de bifurcaciones para todos los if y switch. Esta métrica cuenta el número total de tales ramas en un método y determina el número de ramas ejecutadas o perdidas. La cobertura de bifurcaciones siempre está disponible, incluso en ausencia de información de depuración en los archivos de clase. El manejo de excepciones no se considera ramas en el contexto de esta definición de contador.
 - Sin cobertura: No se han ejecutado ramas en la línea (diamante rojo)
 - Cobertura parcial: Solo se ha ejecutado una parte de los ramales de la línea (diamante amarillo)
 - Cobertura total: se han ejecutado todas las ramas de la línea (diamante verde)

© JMA 2020. All rights reserved

Contadores de cobertura

- Líneas - Lines counters
 - Para todos los archivos de clase que se han compilado con información de depuración, se puede calcular la información de cobertura para líneas individuales. Una línea fuente se considera ejecutada cuando se ha ejecutado al menos una instrucción asignada a esta línea.
 - Sin cobertura: no se ha ejecutado ninguna instrucción en la línea (fondo rojo)
 - Cobertura parcial: solo se ha ejecutado una parte de la instrucción en la línea (fondo amarillo)
 - Cobertura total: se han ejecutado todas las instrucciones de la línea (fondo verde)
- Métodos - Methods counters
 - Cada método no abstracto contiene al menos una instrucción. Un método se considera ejecutado cuando se ha ejecutado al menos una instrucción. Como funciona a nivel de código de bytes, también los constructores y los inicializadores estáticos se cuentan como métodos. Es posible que algunos de estos métodos no tengan una correspondencia directa en el código fuente de Java, como constructores o inicializadores implícitos.
- Clases - Type counters
 - Una clase (o interfaz con código) se considera ejecutada cuando se ha ejecutado al menos uno de sus métodos (incluidos constructores e inicializadores).

© JMA 2020. All rights reserved

Contadores de cobertura

- Complejidad ciclomática
 - También se calcula la complejidad ciclomática para cada método no abstracto y se resume para clases, paquetes y grupos. La complejidad ciclomática es el número mínimo de rutas pasar por todas las instrucciones. Por lo tanto, el valor de complejidad puede servir como una indicación del número de casos de prueba unitarios para cubrir por completo el código.
 - La definición formal de la complejidad ciclomática se basa en la representación del gráfico de flujo de control de un método como un grafo dirigido:
 - Complejidad = [número de aristas] – [número de nodos]+ 2
 - El contador calcula la complejidad ciclomática de un método con la ecuación equivalente:
 - Complejidad = [número de ramas] – [número de puntos de decisión] + 1
 - Según el estado de cobertura de cada bifurcación, JaCoCo también calcula la complejidad cubierta y perdida para cada método. La complejidad perdida nuevamente es una indicación de la cantidad de casos de prueba que faltan para cubrir completamente un módulo. Dado que no se considera el manejo de excepciones como bifurcaciones, los bloques try / catch de las ramas tampoco aumentarán la complejidad.

© JMA 2020. All rights reserved

Calidad de las pruebas

- Se insiste mucho en que la cobertura de test unitarios de los proyectos sea lo más alta posible, pero es evidente que cantidad (de test, en este caso) no siempre implica calidad, la calidad no se puede medir "al peso", y es la calidad lo que realmente importa.
- La cobertura de prueba tradicional (líneas, instrucciones, rama, etc.) mide solo qué código ejecuta las pruebas. No comprueba que las pruebas son realmente capaces de detectar fallos en el código ejecutado, solo pueden identificar el código que no se ha probado.
- Los ejemplos más extremos del problema son las pruebas sin afirmaciones (poco comunes en la mayoría de los casos). Mucho más común es el código que solo se prueba parcialmente, cubrir todo los caminos no implica ejercitarse todos los tipos de equivalencia y valores límite.
- La calidad de las pruebas también debe ser puesta a prueba: No serviría de tener una cobertura del 100% en test unitarios, si no son capaces de detectar y prevenir problemas en el código.
- La herramienta que prueba los test unitarios son los test de mutaciones: Es un test de los test.

© JMA 2020. All rights reserved

Pruebas de mutaciones

- Los pruebas de mutaciones son las pruebas de las pruebas unitarias y el objetivo es tener una idea de la calidad de las pruebas en cuanto a fiabilidad.
- Su funcionamiento es relativamente sencillo: la herramienta que se utilice debe generar pequeños cambios en el código fuente. A estos pequeños cambios se les conoce como mutaciones y crean mutantes.
- Una vez creados los mutantes, se lanzan todos los tests:
 - Si los test unitarios fallan, es que han sido capaces de detectar ese cambio de código. A esto se le llama matar al mutante.
 - Si, por el contrario, los test unitarios pasan, el mutante sobrevive y la fiabilidad (y calidad) de los tests unitarios queda en entredicho.
- Los test de mutaciones presentan informes del porcentaje de mutantes detectados: cuanto más se acerque este porcentaje al 100%, mayor será la calidad de nuestros test unitarios.

© JMA 2020. All rights reserved

Pruebas de mutaciones

- Los mutantes cuyo comportamiento es siempre exactamente igual al del programa original se los llama mutantes funcionalmente equivalentes o, simplemente, mutantes equivalentes, y representan “ruido” que dificulta el análisis de los resultados.
- Para poder matar a un mutante:
 - La sentencia mutada debe estar cubierta por un caso de prueba.
 - Entre la entrada y la salida debe crearse un estado intermedio erróneo.
 - El estado incorrecto debe propagarse hasta la salida.
- La puntuación de mutación para un conjunto de casos de prueba es el porcentaje de mutantes no equivalentes muertos por los datos de prueba:
 - Mutación Puntuación = $100 * D / (N - E)$
donde D es el número de mutantes muertos, N es el número de mutantes y E es el número de mutantes equivalentes

© JMA 2020. All rights reserved

Pitest

- <http://pitest.org/>
- Pitest es un sistema de prueba de mutación de última generación que proporciona una cobertura de prueba estándar de oro para Java y jvm. Es rápido, escalable y se integra con herramientas modernas de prueba y construcción.
- PIT introduce mutaciones en el código (bitcode) y ejecuta las pruebas unitarias contra versiones modificadas automáticamente del código de la aplicación.
- Cuando el código de la aplicación cambia, debería producir resultados diferentes y hacer que las pruebas unitarias fallen. Si una prueba unitaria no falla en esta situación, puede indicar un problema con el conjunto de pruebas.
- La calidad de las pruebas se puede medir a partir del porcentaje de mutaciones muertas.

© JMA 2020. All rights reserved

Pitest

- PIT se puede ejecutar con ant, maven, gradle y otros, o integrarlo en Eclipse, IntelliJ, ...
- La mayoría de los sistemas de prueba de mutaciones para Java son lentos, difíciles de usar y están escritos para satisfacer las necesidades de la investigación académica en lugar de los equipos de desarrollo reales. PIT es rápido: puede analizar en minutos lo que tomaría días en otros sistemas.
- Los informes producidos por PIT están en un formato HTML fácil de leer que combina la cobertura de línea y la información de cobertura de mutación.

© JMA 2020. All rights reserved

BUENAS PRACTICAS

© JMA 2020. All rights reserved

Características de una buena prueba unitaria

- Rápida. No es infrecuente que los proyectos maduros tengan miles de pruebas unitarias. Las pruebas unitarias deberían tardar muy poco tiempo en ejecutarse. Milisegundos.
- Aislada. Las pruebas unitarias son independientes, se pueden ejecutar de forma aislada y no tienen ninguna dependencia en ningún factor externo, como sistemas de archivos o bases de datos.
- Reiterativa. La ejecución de una prueba unitaria debe ser coherente con sus resultados, es decir, devolver siempre el mismo resultado si no cambia nada entre ejecuciones.
- Autocomprobada. La prueba debe ser capaz de detectar automáticamente si el resultado ha sido correcto o incorrecto sin necesidad de intervención humana.
- Oportuna. Una prueba unitaria no debe tardar un tiempo desproporcionado en escribirse en comparación con el código que se va a probar. Si observa que la prueba del código tarda mucho en comparación con su escritura, considere un diseño más fácil de probar.

© JMA 2020. All rights reserved

Asignar nombre a las pruebas

- El nombre de la prueba debe constar de tres partes:
 - Nombre del método que se va a probar.
 - Escenario en el que se está probando.
 - Comportamiento esperado al invocar al escenario.
- Los estándares de nomenclatura son importantes porque expresan de forma explícita la intención de la prueba.

© JMA 2020. All rights reserved

Organizar el código de la prueba

- Prepara, actuar, afirmar es un patrón común al realizar pruebas unitarias. Como el propio nombre implica, consta de tres acciones principales:
 - Prepara los objetos, crearlos y configurarlos según sea necesario.
 - Actuar en un objeto.
 - Afirmar que algo es como se espera.
- Separa claramente en secciones lo que se está probando de los pasos preparación y verificación.
- Las secciones solo deben aparecer una vez como máximo y en el orden establecido.
- Minimiza la posibilidad de mezclar aserciones con el código para "actuar".

© JMA 2020. All rights reserved

Preparación mínima

- La sección de preparación, con la entrada del caso de prueba, debe ser lo más sencilla posible, lo imprescindible para comprobar el comportamiento que se está probando.
- Las pruebas se hacen más resistentes a los cambios futuros en el código base y más cercano al comportamiento de prueba que a la implementación.
- Las pruebas que incluyen más información de la necesaria tienen una mayor posibilidad de incorporar errores en la prueba y pueden hacer confusa su intención. Al escribir pruebas, el usuario quiere centrarse en el comportamiento. El establecimiento de propiedades adicionales en los modelos o el empleo de valores distintos de cero cuando no es necesario solo resta de lo que se quiere probar.

© JMA 2020. All rights reserved

Actuación mínima

- Al escribir las pruebas hay que evitar introducir condiciones lógicas como if, switch, while, for, etc.
- Minimiza la posibilidad de incorporar un error a las pruebas.
- El foco está en el resultado final, en lugar de en los detalles de implementación.
- Al incorporar lógica al conjunto de pruebas, aumenta considerablemente la posibilidad de agregar un error. Cuando se produce un error en una prueba, se quiere saber realmente que algo va mal con el código probado y no en el código que prueba. En caso contrario, restan confianza y las pruebas en las que no se confía no aportan ningún valor.
- El objetivo de la prueba debe ser único, si la lógica en la prueba parece inevitable, denota que el objetivo es múltiple y hay que considerar la posibilidad de dividirla en dos o más pruebas diferentes.

© JMA 2020. All rights reserved

Sustituir literales por constantes

- La asignación de literales a constantes permite dar nombre a los valores, aportando semántica.
- Evita la necesidad de que el lector de la prueba inspeccione el código de producción con el fin de averiguar lo que significa un valor, que hace el valor sea especial.
Assert.IsTrue(rslt.Length <= 10)
- Muestra explícitamente lo que se intenta probar, en lugar de lo que se intenta lograr.
const string VARCHAR_LEN = 10;
Assert.IsTrue(rslt.Length <= VARCHAR_LEN)
- Las valores literales pueden provocar confusión al lector de las pruebas. Si una cadena tiene un aspecto fuera de lo normal, puede preguntarse por qué se ha elegido un determinado valor para un parámetro o valor devuelto. Esto obliga a un vistazo más detallado a los detalles de implementación, en lugar de centrarse en la prueba.

© JMA 2020. All rights reserved

Evitar varias aserciones

- Al escribir las pruebas, hay que intentar incluir solo una aserción por prueba. Los enfoques comunes para usar solo una aserción incluyen:
 - Crear una prueba independiente para cada aserción.
 - Usar pruebas con parámetros.
- Si se produce un error en una aserción, no se evalúan las aserciones posteriores.
- Garantiza que no se estén declarando varios casos en las pruebas.
- Proporciona la imagen exacta de por qué se producen errores en las pruebas.
- Al incorporar varias aserciones en un caso de prueba, no se garantiza que se ejecuten todas. Es un todas o ninguna, se sabe por cual fallo pero no si el resto también falla o es correcto, proporcionando la imagen parcial.
- Una excepción común a esta regla es cuando la validación cubre varios aspectos. En este caso, suele ser aceptable que haya varias aserciones para asegurarse de que el resultado está en el estado que se espera que esté.

© JMA 2020. All rights reserved

Refactorizar código

- La refactorización del código de prueba favorece la reutilización y la legibilidad, simplifican las pruebas.
- Salvo que todos los métodos de prueba usen los mismos requisitos, si se necesita un objeto o un estado similar para las pruebas, es preferible usar métodos auxiliares a los métodos de instalación y desmontaje (si existen):
 - Menos confusión al leer las pruebas, puesto que todo el código es visible desde dentro de cada prueba.
 - Menor posibilidad de configurar más o menos de lo necesario para la prueba.
 - Menor posibilidad de compartir el estado entre las pruebas, lo que crea dependencias no deseadas entre ellas.
- Cada prueba normalmente tendrá requisitos diferentes para funcionar y ejecutarse. Los métodos de instalación y desmontaje son únicos, pero se pueden crear tantos métodos auxiliares como escenarios reutilizables se necesiten.

© JMA 2020. All rights reserved

No validar métodos privados

- En la mayoría de los casos, no debería haber necesidad de probar un método privado.
- Los métodos privados son un detalle de implementación.
- Se puede considerar de esta forma: los métodos privados nunca existen de forma aislada. En algún momento, va a haber un método público que llame al método privado como parte de su implementación. Lo que debería importar es el resultado final del método público que llama al privado.

© JMA 2020. All rights reserved

Aislar las pruebas

- Las dependencias externas afectan a la complejidad de la estrategia de pruebas, hay que aislar a las pruebas de las dependencias externas, sustituyendo las dependencias por dobles de prueba, salvo que se este probando específicamente dichas dependencias.
- Siguiendo la misma regla de oro, las pruebas de integración y sistema deben estar aisladas de sus dependencias salvo cuando se estén probando dichas dependencias. Así mismo, el resultado de las pruebas debe ser previsible.
- Entre las ventajas de esta aproximación se encuentran:
 - Devuelven resultados determinísticos
 - Permiten crear o reproducir determinados estados (por ejemplo errores de conexión)
 - Obtienen resultados mucho mas rápidamente y a menor coste, incluso offline.
 - Permiten el inicio temprano de las pruebas incluso cuando las dependencias todavía no están disponibles.
 - Permiten incluir atributos o métodos exclusivamente para el testeo.

© JMA 2020. All rights reserved

Cubrir aspectos no evidentes

- Las pruebas no deben cubrir solo los casos evidentes, los correctos, sino que deben ampliarse a los casos incorrectos.
- Un juego de pruebas debe ejercitar la resiliencia: la capacidad de resistir los errores y la recuperación ante los mismos.
- En los cálculos no hay que comprobar solamente si realiza correctamente el calculo, también hay que verificar que es el calculo que se debe realizar.
- Los dominios de los datos determinan la validez de los mismos y fijan la calidad de la información, dichos dominios deben ser ejercitados profundamente.

© JMA 2020. All rights reserved

Respetar los límites de las pruebas

- Las pruebas unitarias ejercitan profundamente los componentes de forma aislada centrándose en la funcionalidad, los cálculos, las reglas de dominio y semánticas de los datos. Opcionalmente la estructura del código, es decir, sentencias, decisiones, bucles y caminos distintos.
- Las pruebas de integración se basan en componentes ya probados (unitaria o integración) o en dobles de pruebas y se centran en la estructura de llamadas, secuencias o colaboración, así como en la transición de estados.
- Hay muchos tipos de pruebas de sistema y cada uno pone el foco en un aspecto muy concreto, cada prueba solo debe cubrir un solo aspecto. Las pruebas funcionales del sistema son las pruebas de integración de todo el sistema centrándose en compleción de la funcionalidades y los procesos de negocio, su estructura, disponibilidad y accesibilidad.

© JMA 2020. All rights reserved



Docker

<https://www.docker.com>

© JMA 2020. All rights reserved

INTRODUCCIÓN E INSTALACIÓN

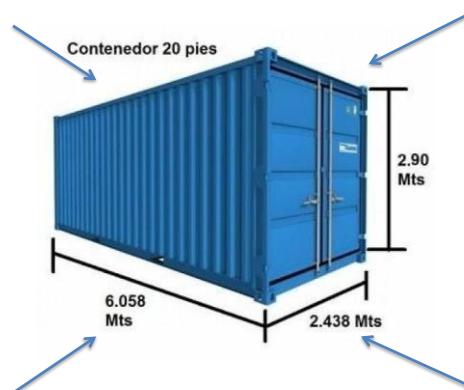
© JMA 2020. All rights reserved

Antecedentes

- El transporte de contenedores fue creado en 1956 por Malcolm MacLean, un transportista de carretera de Carolina del Norte. En 1953, este empresario se dio cuenta de que las carreteras que conectan los diversos puertos de la costa oeste estaban completamente saturadas y le surgió la idea de cargar directamente remolques de camiones en los barcos.
- Un contenedor marítimo es una gran caja de metal de dimensiones estándar usada para transportar mercancías.
- Fue ideado especialmente para el transporte sin manipulación intermedia o ruptura de la carga. Asimismo fue pensado para poder ser versátil al ser desplazado por cualquier medio de transporte (por carretera, marítimo, ferroviario) teniendo en cuenta la combinación de varios de ellos.
- Un contenedor marítimo es apilable, maniobrable, apto para dispositivos ad-hoc y pensado para un uso intensivo. Por estas razones se ha convertido en una herramienta indispensable para la logística hoy en día y la globalización.

© JMA 2020. All rights reserved

Antecedentes



© JMA 2020. All rights reserved

Antecedentes



© JMA 2020. All rights reserved

Contenedores

- Del mismo modo que los contenedores de mercancías permiten su transporte por barco, tren o camión independientemente de la carga de su interior, los contenedores de software actúan como una unidad estándar de implementación de software que puede contener diferentes dependencias y código. De esta manera, la inclusión del software en contenedor permite a los desarrolladores y los profesionales de TI implementarlo en entornos con pocas modificaciones o ninguna en absoluto.
- La inclusión en contenedores es un enfoque de desarrollo de software en el que una aplicación o un servicio, sus dependencias y su configuración (extraídos como archivos de manifiesto de implementación) se empaquetan como una imagen de contenedor. La aplicación en contenedor puede probarse como una unidad e implementarse como una instancia de imagen de contenedor en el sistema operativo (SO) host.
- Los contenedores también aíslan las aplicaciones entre sí en un sistema operativo compartido. Las aplicaciones en contenedor se ejecutan sobre un host de contenedor que a su vez se ejecuta en el sistema operativo (Linux o Windows). Por lo tanto, los contenedores tienen una superficie significativamente menor que las imágenes de máquina virtual (VM).
- Otra ventaja de la inclusión en contenedores es la escalabilidad. La creación de contenedores para tareas a corto plazo permite escalar horizontalmente con gran rapidez. Desde el punto de vista de la aplicación, la creación de instancias de una imagen (la creación de un contenedor) es similar a la creación de instancias de un proceso como un servicio o una aplicación web. Pero con fines de confiabilidad, cuando ejecute varias instancias de la misma imagen en varios servidores host, seguramente le interesará que cada contenedor (instancia de imagen) se ejecute en un servidor host o máquina virtual diferente en dominios de error distintos.

© JMA 2020. All rights reserved

Modelo de despliegue

- El modelo de despliegue hace referencia al modo en que vamos a organizar y gestionar los despliegues de las aplicaciones y los servicios, así como a las tecnologías que podemos usar para tal fin.
- Existen convencionalmente varios patrones en este sentido a la hora de encapsular y desplegar:
 - Máquinas virtuales.
 - Contenedores.
 - Sin servidor: FaaS (Functions-as-a-Service)
- Las máquinas virtuales (VM) son una abstracción del hardware físico que convierte un servidor en muchos servidores. El hipervisor permite que varias máquinas virtuales se ejecuten en una sola máquina. Cada máquina virtual incluye una copia completa de un sistema operativo, la aplicación, los archivos binarios y las bibliotecas necesarios, lo que ocupa decenas de GB. Las máquinas virtuales también pueden tardar en arrancar.
- Las máquinas virtuales incluyen la aplicación, las bibliotecas o los archivos binarios necesarios y un sistema operativo invitado completo. Para las máquinas virtuales, hay tres niveles de base en el servidor host, de manera ascendente: infraestructura, sistema operativo host y un hipervisor y, encima de todo eso, cada máquina virtual tiene su propio sistema operativo y todas las bibliotecas necesarias.

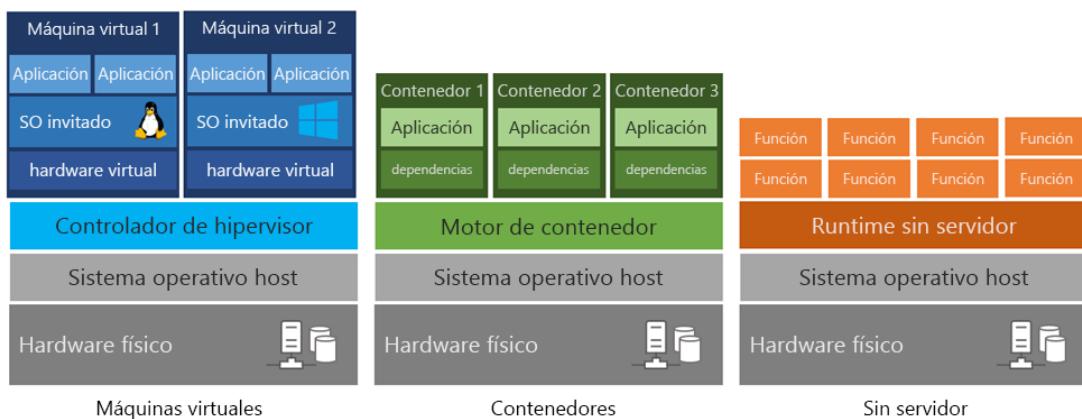
© JMA 2020. All rights reserved

FaaS (Functions-as-a-Service)

- El auge de la informática sin servidor es una de las innovaciones más importantes de la actualidad. Las tecnologías sin servidor, como Azure Functions, AWS Lambda o Google Cloud Functions, permiten a los desarrolladores centrarse por completo en escribir código. Toda la infraestructura informática de la que dependen (máquinas virtuales (VM), compatibilidad con la escalabilidad y demás) se administra por ellos. Debido a esto, la creación de aplicaciones se vuelve más rápida y sencilla. Ejecutar dichas aplicaciones a menudo resulta más barato, porque solo se le cobra por los recursos informáticos que realmente usa el código.
- La arquitectura serverless habilita la ejecución de una aplicación mediante contenedores efímeros y sin estado; estos son creados en el momento en el que se produce un evento que dispare dicha aplicación. Contrariamente a lo que nos sugiere el término, serverless no significa «sin servidor», sino que éstos se usan como un elemento anónimo más de la infraestructura, apoyándose en las ventajas del cloud computing.
- La tecnología sin servidor apareció por primera vez en lo que se conoce como tecnologías de plataforma de aplicaciones como servicio (aPaaS), actualmente como FaaS (Functions-as-a-Service).

© JMA 2020. All rights reserved

Contenedores



© JMA 2020. All rights reserved

Docker

- Docker es un proyecto de código abierto para automatizar la implementación de aplicaciones como contenedores portátiles y autosuficientes que se pueden ejecutar en la nube o localmente. Docker es también una empresa que promueve e impulsa esta tecnología, en colaboración con proveedores de la nube, Linux y Windows.
- Salomon Hykes comenzó Docker como un proyecto interno dentro de dotCloud, empresa enfocada a PaaS (plataforma como servicio). Fue liberado como código abierto en marzo de 2013.
- Aprovechó los conceptos informáticos existentes en torno a los contenedores y, específicamente, en el mundo de Linux, las primitivas conocidas como cgroups y espacios de nombres.
- El éxito en el mundo de Linux impulsó una asociación con Microsoft que llevó los contenedores Docker y su funcionalidad a Windows Server.
- La tecnología disponible de Docker y su proyecto de código abierto, Moby, ha sido aprovechada por todos los principales proveedores de centros de datos y de nube. Muchos de estos proveedores están aprovechando Docker para ofrecer IaaS de contenedores-nativos. Además, los principales marcos sin servidor de código abierto utilizan la tecnología de contenedores Docker.
- En junio de 2015, Docker donó la especificación de la imagen del contenedor y el código en tiempo de ejecución, ahora conocido como runc, a Open Container Initiative (OCI) para ayudar a establecer la estandarización a medida que el ecosistema de contenedores crece y madura.

© JMA 2020. All rights reserved

Contenedores Docker

- Un contenedor es una unidad estándar de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable de un entorno informático a otro. Una imagen de contenedor de Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones.
- Las imágenes de contenedores se convierten en contenedores en tiempo de ejecución y, en el caso de los contenedores de Docker, las imágenes se convierten en contenedores cuando se ejecutan en Docker Engine. Disponible para aplicaciones basadas en Linux y Windows, el software en contenedores siempre se ejecutará de la misma manera, independientemente de la infraestructura. Los contenedores aíslan el software de su entorno y garantizan que funcione de manera uniforme a pesar de las diferencias, por ejemplo, entre el desarrollo y la puesta en escena.
- Contenedores Docker que se ejecutan en Docker Engine:
 - Estándar: Docker creó el estándar de la industria para contenedores.
 - Ligero: los contenedores comparten el kernel del sistema operativo de la máquina y, por lo tanto, no requieren un sistema operativo por aplicación, lo que proporciona una mayor eficiencia del servidor, reduciendo los costes del servidor y las licencias.
 - Seguro: las aplicaciones son más seguras en contenedores y Docker proporciona las capacidades de aislamiento predeterminadas más sólidas de la industria.

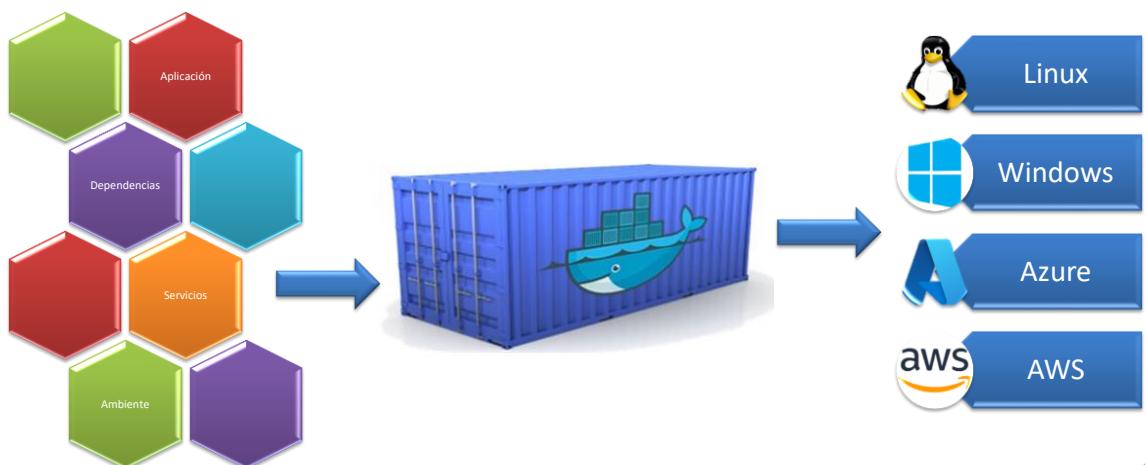
© JMA 2020. All rights reserved

Entornos

- Los contenedores de Docker se pueden ejecutar en cualquier lugar, a nivel local en el centro de datos de cliente, en un proveedor de servicios externo o en la nube. Los contenedores de imagen de Docker se pueden ejecutar de forma nativa en Linux y Windows. Sin embargo, las imágenes de Windows solo pueden ejecutarse en hosts de Windows y las imágenes de Linux pueden ejecutarse en hosts de Linux y hosts de Windows (con una máquina virtual Linux de Hyper-V o con WSL 2, Subsistema de Windows para Linux o Linux en Windows), donde host significa un servidor o una máquina virtual.
- Los desarrolladores pueden usar entornos de desarrollo en Windows, Linux o macOS. En el equipo de desarrollo, el desarrollador ejecuta un host de Docker en que se implementan imágenes de Docker, incluidas la aplicación y sus dependencias. Los desarrolladores que trabajan en Linux o macOS usan un host de Docker basado en Linux solo pueden crear imágenes para contenedores de Linux. Los desarrolladores que trabajan en Windows pueden crear imágenes para contenedores de Windows o Linux.
- Para hospedar contenedores en entornos de desarrollo y disponer de las herramientas de desarrollo, Docker suministra Docker Desktop para Windows o para macOS. Estos productos instalan la máquina virtual necesaria (el host de Docker) para hospedar los contenedores.

© JMA 2020. All rights reserved

Despliegue



© JMA 2020. All rights reserved

Ventajas y Desventajas

- Usar contenedores Docker permite a desarrolladores y administradores de sistemas desarrollar y probar aplicaciones o servicios en un entorno seguro e igual al de producción.
- Las principales ventajas de usar contenedores Docker son:
 - Las instancias se inician en pocos segundos.
 - Son fácilmente replicables.
 - Es fácil de automatizar e integrar en entornos CI (integración continua).
 - Consumen menos recursos que las máquinas virtuales tradicionales.
 - Tienen un mayor rendimiento que la virtualización tradicional ya que corre directamente sobre el Kernel de la máquina en la que se aloja, evitando al hipervisor.
 - Ocupan mucho menos espacio.
 - Permiten aislar las dependencias de una aplicación de las instaladas en el host.
 - Existe un gran repositorio de imágenes ya creadas sobre miles de aplicaciones, que además pueden modificarse libremente.
- Las principales desventajas de usar contenedores Docker son:
 - Sólo puede usarse de forma nativa en entornos Unix con Kernel igual o superior a 3.8.
 - Sólo soporta arquitecturas de 64 bits.

© JMA 2020. All rights reserved

Uso y Recomendaciones

- Entrega rápida y consistente de las aplicaciones
 - Docker agiliza el ciclo de vida del desarrollo al permitir que los desarrolladores trabajen en entornos estandarizados utilizando contenedores locales que proporcionan sus aplicaciones y servicios. Los contenedores son excelentes para los flujos de trabajo de integración continua y entrega continua (CI/CD).
- Despliegue adaptativo y escalado
 - La plataforma basada en contenedores de Docker permite cargas de trabajo altamente portátiles. Los contenedores Docker pueden ejecutarse en un equipo portátil local de un desarrollador, en máquinas físicas o virtuales en un centro de datos, en proveedores de la nube o en una combinación de entornos. La portabilidad y la naturaleza liviana de Docker también facilitan la administración dinámica de cargas de trabajo, ampliando o eliminando aplicaciones y servicios según lo dicten las necesidades comerciales, casi en tiempo real.
- Ejecución de mayores cargas de trabajo en el mismo hardware
 - Docker es ligero y rápido. Proporciona una alternativa viable y rentable a las máquinas virtuales basadas en hipervisor, por lo que puede utilizar una mayor parte de su capacidad de cómputo para lograr sus objetivos comerciales. Docker es perfecto para implementaciones pequeñas, medianas y entornos de alta densidad que necesitan hacer más con menos recursos.

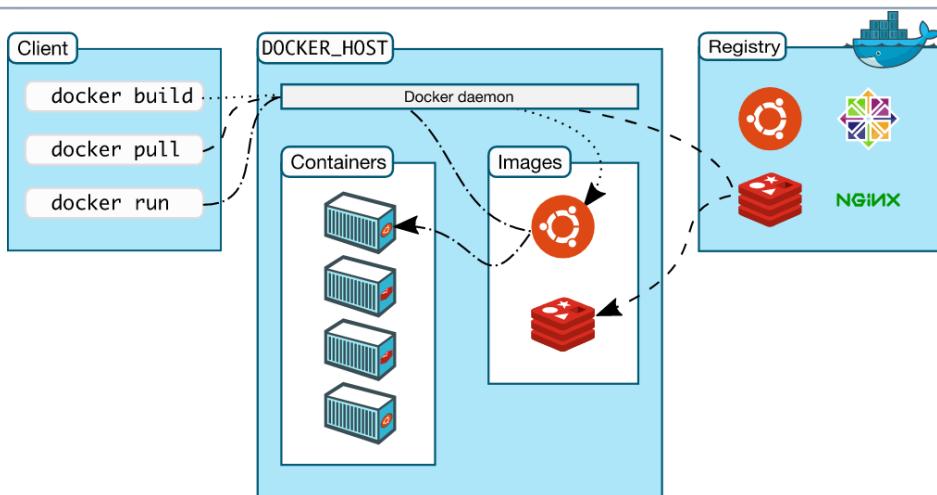
© JMA 2020. All rights reserved

Arquitectura

- Docker utiliza una arquitectura cliente-servidor.
- El cliente de Docker se comunica con el demonio de Docker (servicio), que hace el trabajo pesado de crear, ejecutar y distribuir los contenedores.
- El cliente y el demonio de Docker pueden ejecutarse en el mismo sistema o se puede conectar un cliente de Docker a un demonio de Docker remoto. El cliente Docker y el demonio se comunican mediante una API REST, a través de sockets UNIX o una interfaz de red.
- Otro cliente de Docker es Docker Compose, que permite trabajar con aplicaciones que requieren un conjunto de contenedores.
- Un registro de Docker almacena imágenes de Docker. Docker Hub es un registro público que cualquiera puede usar y Docker está configurado para buscar imágenes en Docker Hub de manera predeterminada.

© JMA 2020. All rights reserved

Arquitectura



© JMA 2020. All rights reserved

Componentes

- El demonio Docker
 - El demonio de Docker (dockerd) escucha las solicitudes de la API de Docker y administra los objetos de Docker, como imágenes, contenedores, redes y volúmenes. Un demonio también puede comunicarse con otros demonios para administrar los servicios de Docker.
- El cliente Docker
 - El cliente de Docker (docker) es la forma principal en que muchos usuarios de Docker interactúan con Docker. Cuando se utilizan comandos como docker run, el cliente envía estos comandos a dockerd, que los lleva a cabo. El comando Docker utiliza la API de Docker. El cliente de Docker puede comunicarse con más de un daemon.
- Docker Desktop
 - Docker Desktop es una aplicación fácil de instalar para entornos Mac o Windows que permite crear y compartir microservicios y aplicaciones en contenedores. Docker Desktop incluye el demonio Docker (dockerd), el cliente Docker (docker), Docker Compose, Docker Content Trust, Kubernetes y Credential Helper.

© JMA 2020. All rights reserved

Conceptos

- **Imagen:** un paquete con todas las dependencias y la información necesarias para crear un contenedor. Una imagen incluye todas las dependencias, así como la configuración de implementación y ejecución que usará el runtime de un contenedor. Normalmente, una imagen se deriva de varias imágenes base que son capas que se apilan unas encima de otras para formar el sistema de archivos del contenedor. Una vez que se crea una imagen, esta es inmutable.
- **Dockerfile:** archivo de texto que contiene instrucciones sobre cómo compilar una imagen de Docker. Es como un script por lotes; la primera línea indica la imagen base con la que se comienza y, después, deben seguirse las instrucciones para instalar programas necesarios, copiar archivos, etc., hasta obtener el entorno de trabajo que se necesita.
- **Compilación:** la acción de crear una imagen de contenedor basada en la información y el contexto que proporciona su Dockerfile, así como archivos adicionales en la carpeta en que se crea la imagen.
- **Etiqueta:** una marca o una etiqueta que se puede aplicar a las imágenes para que se puedan identificar diferentes imágenes o versiones de la misma imagen (según el número de versión o el entorno de destino).

© JMA 2020. All rights reserved

Conceptos

- **Contenedor:** una instancia de una imagen de Docker. Un contenedor representa la ejecución de una sola aplicación, proceso o servicio. Está formado por el contenido de una imagen de Docker, un entorno de ejecución y un conjunto estándar de instrucciones. Al escalar un servicio, crea varias instancias de un contenedor a partir de la misma imagen. O bien, un proceso por lotes puede crear varios contenedores a partir de la misma imagen y pasar parámetros diferentes a cada instancia. Los datos persistentes se pierden con el contenedor.
- **Volúmenes:** ofrece un sistema de archivos persistente que el contenedor puede usar. Puesto que las imágenes son de solo lectura pero la mayoría de los programas necesitan escribir en el sistema de archivos, los volúmenes agregan una capa grabable, encima de la imagen de contenedor, por lo que los programas tienen acceso a un sistema de archivos grabable. Es transparente para la aplicación que no sabe que está accediendo a un sistema de archivos por capas. Los volúmenes residen en el sistema host y los administra Docker.

© JMA 2020. All rights reserved

Conceptos

- **Repositorio:** una colección de imágenes de Docker relacionadas, etiquetadas con una etiqueta que indica la versión de la imagen. Algunos repositorios contienen varias variantes de una imagen específica, como una imagen que contiene SDK (más pesada), una imagen que solo contiene runtimes (más ligera), etcétera. Estas variantes se pueden marcar con etiquetas. Un solo repositorio puede contener variantes de plataforma, como una imagen de Linux y una imagen de Windows.
- **Registro:** servicio que proporciona acceso a los repositorios. El registro predeterminado para la mayoría de las imágenes públicas es Docker Hub (propiedad de la empresa Docker). Normalmente, un registro contiene repositorios procedentes de varios equipos. Las empresas suelen tener registros privados para almacenar y administrar imágenes que han creado.

© JMA 2020. All rights reserved

Conceptos

- **Docker Hub:** registro público para cargar imágenes y trabajar con ellas. Docker Hub proporciona hospedaje de imágenes de Docker, registros públicos o privados, desencadenadores de compilación y enlaces web e integración con GitHub y Bitbucket.
- **Azure Container Registry (ACI), Amazon Elastic Container Service (ECS):** recurso públicos para trabajar con imágenes de Docker y sus componentes en Azure y en AWS. Esto proporciona un registro cercano a las implementaciones en Azure o AWS proporcionando control sobre el acceso, lo que permite usar los grupos y los permisos.
- **Docker Trusted Registry (DTR):** servicio del registro de Docker (ofrecido por Docker) que se puede instalar de forma local, por lo que se encuentra en el centro de datos y la red de la organización. Es ideal para imágenes privadas que deben administrarse dentro de la empresa. Docker Trusted Registry se incluye como parte del producto Docker Datacenter.

© JMA 2020. All rights reserved

Conceptos

- **Clúster:** colección de hosts de Docker que se expone como si fuera un solo host de Docker virtual, de manera que la aplicación se puede escalar a varias instancias de los servicios repartidos entre varios hosts del clúster. Los clústeres de Docker se pueden crear con Kubernetes, Docker Swarm, ...
- **Orquestador:** herramienta que simplifica la administración de clústeres y hosts de Docker. Los orquestadores permiten administrar las imágenes, los contenedores y los hosts a través de una interfaz de la línea de comandos (CLI) o una interfaz gráfica de usuario. Puede administrar las redes de contenedor, las configuraciones, el equilibrio de carga, la detección de servicios, la alta disponibilidad, la configuración del host de Docker y muchas cosas más. Un orquestador se encarga de ejecutar, distribuir, escalar y reparar las cargas de trabajo a través de una colección de nodos.

© JMA 2020. All rights reserved

Conceptos

- **Compose:** herramienta de línea de comandos y formato de archivo YAML con metadatos para definir y ejecutar aplicaciones de varios contenedores. Primero se define una sola aplicación basada en varias imágenes con uno o más archivos .yml que pueden invalidar los valores según el entorno. Después de crear las definiciones, puede implementar toda la aplicación de varios contenedores con un solo comando (docker-compose up) que crea un contenedor por imagen en el host de Docker.
- **Compilación de varias fases:** es una característica, desde Docker 17.05 o versiones posteriores, que ayuda a reducir el tamaño de las imágenes finales. Por ejemplo, se puede usar una imagen base grande que contenga el SDK para compilar y publicar y una imagen base pequeña solo de tiempo de ejecución para hospedar la aplicación.
- **Imagen multiarquitectura:** En el caso de la arquitectura múltiple, es una característica que simplifica la selección de la imagen adecuada, según la plataforma donde se ejecuta Docker. Por ejemplo, si un Dockerfile solicita una imagen base FROM mcr.microsoft.com/dotnet/sdk:6.0 del Registro, en realidad obtendrá 6.0-nanoserver-20H2, 6.0-nanoserver-1809 o 6.0-bullseye-slim, según el sistema operativo en el que se ejecute Docker y la versión.

© JMA 2020. All rights reserved

Instalación

- Se puede descargar e instalar Docker en varias plataformas:
 - Docker Desktop en Linux
 - Ubuntu
 - Debian
 - Fedora
 - Docker Desktop en Windows
 - WSL 2 backend
 - Hyper-V backend and Windows containers
 - Docker Desktop en Mac
 - Mac with Intel chip
 - Mac with Apple silicon (arm64)
- Dado que las instalaciones varían en función a las distribuciones y versión es recomendable consultar siempre la documentación oficial:
 - <https://docs.docker.com/get-docker/>

© JMA 2020. All rights reserved

GUÍA RÁPIDA

© JMA 2020. All rights reserved

Introducción

- Docker Engine es una tecnología de contenedores de código abierto para crear y contener las aplicaciones. Docker Engine actúa como una aplicación cliente-servidor con:
 - Un servidor con un proceso daemon dockerd de ejecución prolongada.
 - API que especifican interfaces que los programas pueden usar para hablar e instruir al demonio Docker.
 - Un cliente de interfaz de línea de comandos (CLI) docker.
- La CLI usa las API de Docker para controlar o interactuar con el demonio de Docker a través de secuencias de comandos o comandos directos de la CLI. Muchas otras aplicaciones de Docker utilizan la API y la CLI subyacentes. El daemon crea y administra objetos Docker, como imágenes, contenedores, redes y volúmenes.
- Docker Dashboard brinda un GUI para contenedores, imágenes y volúmenes. Docker Dashboard está disponible para Mac, Windows y Linux.

© JMA 2020. All rights reserved

Crear un contenedor

- Para crear un contenedor partiendo de la imagen de ejemplo, abra un símbolo del sistema o una ventana bash y ejecute el comando:
 - docker run -d -p 8080:80 --name tutorial docker/getting-started
- El comando run primero crea un contenedor sobre la imagen especificada y luego lo arranca. La primera vez debe descargarse la imagen y cachearla en el repositorio, lo que consume su tiempo. Los parámetros le indican:
 - d Ejecutar el contenedor en modo separado (en segundo plano, liberando la consola) y muestra el identificador del contenedor.
 - p 8080:80 Asignar el puerto 8080 del host al puerto 80 en el contenedor.
 - name tutorial Asignar un nombre al contenedor.
 - docker/getting-started Especificar la imagen a utilizar.
- Para ver los contenedores en ejecución:
 - docker ps
- Para acceder a la aplicación web del contenedor, vaya a <http://localhost:8080> en un navegador web.
- Para eliminar el contenedor forzando a que lo pare:
 - docker rm -f tutorial

© JMA 2020. All rights reserved

¿Qué es un contenedor?

- Un contenedor es un proceso en su máquina que está aislado de todos los demás procesos en la máquina host. El aislamiento aprovecha los espacios de nombres del kernel y cgroups, características que han estado en Linux durante mucho tiempo. Docker ha trabajado para que estas capacidades sean accesibles y fáciles de usar.
- En resumen, un contenedor:
 - es una instancia ejecutable de una imagen. Puede crear, iniciar, detener, mover o eliminar un contenedor mediante el DockerAPI o el CLI.
 - se puede ejecutar en máquinas locales, máquinas virtuales o implementarse en la nube.
 - es portátil (se puede ejecutar en cualquier sistema operativo).
 - está aislado de otros contenedores y ejecuta su propio software, archivos binarios y configuraciones.

© JMA 2020. All rights reserved

¿Qué es una imagen de contenedor?

- Cuando se ejecuta un contenedor, utiliza un sistema de archivos aislado. Este sistema de archivos personalizado lo proporciona una imagen de contenedor.
- Dado que la imagen contiene el sistema de archivos del contenedor, debe contener todo lo necesario para ejecutar una aplicación: todas las dependencias, configuraciones, scripts, archivos binarios, etc.
- La imagen también contiene otra configuración para el contenedor, como variables de entorno, un comando predeterminado para ejecutar, y otros metadatos.
- Básicamente, la imagen es la plantilla con la que se crean los contenedores.

© JMA 2020. All rights reserved

Crea la imagen del contenedor

- Para construir la aplicación, necesitamos usar un archivo Dockerfile. Un Dockerfile es simplemente un script de instrucciones basado en texto que se usa para crear una imagen de contenedor.
- Crear un archivo con el nombre Dockerfile en la misma carpeta que la aplicación:

```
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY ..
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

© JMA 2020. All rights reserved

Crea el contenedor

- Para crear la imagen del contenedor y cachearla en el repositorio se usa el comando:
 - docker build -t getting-started .
- Es posible que se tengan que descargar muchas "capas". El -t establece la etiqueta de la imagen, un nombre legible. El . al final del comando le dice a Docker que debe buscar el Dockerfile en el directorio actual.
- Para ver las imágenes locales:
 - docker images
- Para crear un contenedor con la imagen que acabamos de crear:
 - docker run -dp 3000:3000 --name tutorial-app getting-started
- La creación es muy rápida porque ya se dispone de la imagen localmente.
- Despues de unos segundos, navegar a <http://localhost:3000>. Se debería ver la aplicación.
- Agregar uno o dos elementos y vea que funciona como espera. Puede marcar elementos como completos o eliminarlos. La interfaz está almacenando con éxito los elementos en el backend.
- Para parar y arrancar el contenedor:
 - docker stop tutorial-app
 - docker start tutorial-app

© JMA 2020. All rights reserved

Actualizar la aplicación

- Modificar la aplicación.
- Construir la nuestra versión actualizada de la imagen, usando el mismo comando: si no existe la crea, si existe la reemplaza:
 - docker build -t getting-started .
- El contenedor es una instancia que se creo con una plantilla (imagen), no es posible cambiar la imagen con la que se creo un contenedor. Para cambiar la imagen del contenedor hay que reemplazarlo: parar y destruir el contenedor anterior, crear el contenedor con la versión actualizada de la imagen:
 - docker stop tutorial-app
 - docker rm tutorial-app
 - docker run -dp 3000:3000 --name tutorial-app getting-started
- Al navegar a <http://localhost:3000> se debería ver la aplicación, pero los datos se han perdido con el contenedor borrado.

© JMA 2020. All rights reserved

Volúmenes

- Cuando se ejecuta un contenedor, utiliza las distintas capas de una imagen para su sistema de archivos. Cada contenedor también tiene su propio "espacio borrador" para crear/actualizar/eliminar archivos y cualquier cambio no se verá en otro contenedor, incluso si están usando la misma imagen.
 - docker run -dp 3001:3000 --name tutorial-app2 getting-started
- Los volúmenes brindan la capacidad de conectar rutas específicas del sistema de archivos del contenedor a la máquina host. Si se monta un directorio en el contenedor, los cambios en ese directorio también se ven en la máquina host y si montamos ese mismo directorio en los reinicios del contenedor, veríamos los mismos archivos.
- Para crear un volumen con nombre:
 - docker volume create todo-db
- Para crear un contenedor que utilice el volumen con nombre:
 - docker run -dp 3000:3000 -v todo-db:/etc/todos --name tutorial-app getting-started
- Si se recrea el contenedor apuntando al mismo volumen debería conservar los datos.
- Para ver los detalles del volumen:
 - docker volume inspect todo-db

© JMA 2020. All rights reserved

Redes de contenedores

- Las aplicaciones complejas requieren múltiples servicios: bases de datos, colas de mensajes, ... Cada contenedor debe hacer una sola cosa y hacerla bien:
 - Es muy probable que se tenga que escalar las API y los front-end de manera diferente a las bases de datos y otros servicios.
 - Los contenedores separados permiten versionar y actualizar versiones de forma aislada.
 - Si bien puede usar un contenedor para la base de datos localmente, es posible que se desee usar un servicio administrado para la base de datos en producción, en cuyo caso no se desea enviar el motor de base de datos con la aplicación.
 - La ejecución de varios procesos requeriría un administrador de procesos (el contenedor solo inicia un proceso), lo que agrega complejidad al inicio/apagado del contenedor.
- Los contenedores, de forma predeterminada, se ejecutan de forma aislada y no saben nada sobre otros procesos o contenedores en la misma máquina. La creación de redes permite que si dos contenedores están en la misma red, puedan comunicarse entre sí.

© JMA 2020. All rights reserved

Redes de contenedores

- Para crear una red:
 - docker network create todo-net
- Para crear un contenedor MySQL y adjuntarlo a la red:
 - docker run -d --network todo-net --network-alias mysql -p 3306:3306 -v todo-mysql-data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=todos --name tutorial-mysql mysql:5.7
- Para crear el contenedor de la aplicación en la misma red y configurar la conexión a la base de datos:
 - docker run -dp 3001:3000 --network todo-net -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=root -e MYSQL_DB=todos --name tutorial-db getting-started

© JMA 2020. All rights reserved

Usar composición

- Docker Compose es una herramienta que se desarrolló para ayudar a definir y compartir aplicaciones de varios contenedores. Con Compose, podemos crear un archivo YAML para definir los servicios y, con un solo comando, podemos activar o desactivar todo.
- La gran ventaja de usar Compose es que puede definir la pila de su aplicación en un archivo, mantenerla en la raíz del repositorio de su proyecto (ahora tiene control de versión) y permitir fácilmente que otra persona contribuya a su proyecto. Alguien solo necesitaría clonar su repositorio e iniciar la aplicación de redacción. De hecho, es posible que vea bastantes proyectos en GitHub/GitLab haciendo exactamente esto ahora.
- En la raíz del proyecto de la aplicación, se crea un archivo llamado docker-compose.yml.
- El nuevo Compose V2 como comando compose del CLI docker, sustituye al CLI docker-compose anterior.

© JMA 2020. All rights reserved

docker-compose.yml

```
version: "3.7"

services:
  mysql:
    image: mysql:5.7
    ports:
      - 3306:3306
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: todos
    networks:
      - todo-net
  app:
    image: getting-started
    ports:
      - 3000:3000
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: root
      MYSQL_DB: todos
    networks:
      - todo-net
    volumes:
      todo-mysql-data:
    networks:
      todo-net:
```

© JMA 2020. All rights reserved

docker compose

- En el directorio que contiene el fichero docker-compose.yml, con una consola, se ejecuta el comando (con -d para ejecutar todo en segundo plano):
 - docker compose up -d
- El comando crea en un solo paso los dos contenedores, la red y el volumen.
- Para ver los registros de los contenedores (-f dará una salida en vivo a medida que se genera):
 - docker compose logs -f
- En este punto, se debería poder abrir la aplicación y verla ejecutándose.
- Para eliminar todo en un único paso: los dos contenedores, la red y el volumen (de forma predeterminada, los volúmenes con nombre no se eliminan salvo que se agregue --volumes) se ejecuta el comando:
 - docker compose down --volumes

© JMA 2020. All rights reserved

Registro

- Para compartir imágenes de Docker, debe usar un registro de Docker. El registro predeterminado es Docker Hub y es de donde provienen todas las imágenes que hemos usado.
 - <https://hub.docker.com/>
- Los usuarios obtienen acceso a repositorios públicos gratuitos para almacenar y compartir imágenes o pueden elegir un plan de suscripción para repositorios privados.
- Para enviar una imagen, primero debemos crear una cuenta en Docker Hub. Las cuentas publicas son gratuitas.
- Para subir (empujar) una imagen al repositorio:
 - Iniciar sesión en Docker Hub en la consola:
 - docker login -u YOUR-USER-NAME
 - Asignar a la imagen una etiqueta con el nombre del repositorio que incluye el nombre de usuario y la versión (latest por defecto):
 - docker tag getting-started YOUR-USER-NAME/getting-started:1.0
 - Empujar o instalar la imagen en el servidor de registro:
 - docker push YOUR-USER-NAME/getting-started:1.0
 - Iniciar sesión en el sitio web de Docker Hub con el navegador y completar la documentación del repositorio.

© JMA 2020. All rights reserved

IMÁGENES

© JMA 2020. All rights reserved

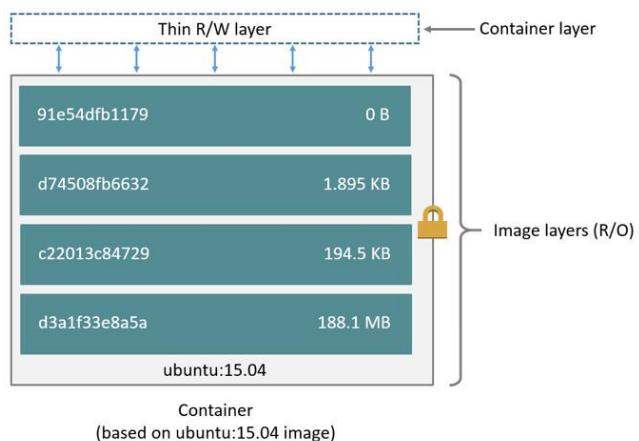
Capas

- Las imágenes de Docker son plantillas de solo lectura desde las que se lanzan los contenedores de Docker. Cada imagen consta de una serie de capas. Docker utiliza el sistema de archivos de unión de Linux para combinar estas capas en una sola imagen. Los sistemas de archivos de unión permiten que los archivos y directorios de sistemas de archivos separados, conocidos como ramas, se superpongan de forma transparente, formando un único sistema de archivos superpuestos coherente.
- Cada capa representa una instrucción en el Dockerfile de la imagen. Cada capa, excepto la última, es de solo lectura. Cada capa es solo un conjunto de diferencias con respecto a la capa anterior. Hay que tener en cuenta que tanto modificar como eliminar archivos dará como resultado una nueva capa, pero aún estarán disponibles en la capa anterior y se sumará al tamaño total de la imagen.
- Las capas se apilan una encima de la otra. Cuando se crea un nuevo contenedor, se agrega una nueva capa de escritura encima de las capas subyacentes. Esta capa a menudo se denomina "capa contenedora". Todos los cambios realizados en el contenedor en ejecución, como la escritura de nuevos archivos, la modificación de archivos existentes y la eliminación de archivos, se escriben en esta fina capa contenedora de escritura.

© JMA 2020. All rights reserved

Capas

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
RUN rm -r $HOME/.cache
CMD python /app/app.py
```



© JMA 2020. All rights reserved

Capas

- Una de las razones por las que Docker es tan liviano es por estas capas. Cuando cambia una imagen de Docker se crea una nueva capa. Por lo tanto, en lugar de reemplazar toda la imagen o reconstruirla por completo, como hace con una máquina virtual, solo se agrega o actualiza esa capa.
- Cada imagen parte de una imagen base, por ejemplo ubuntu, una imagen base de Ubuntu. También se pueden usar nuestras propias imágenes como base para una nueva imagen, por ejemplo, crear una imagen base de Apache para usarla como base de todas las imágenes de su aplicación web.
- Las imágenes de Docker se construyen a partir de estas imágenes base mediante un conjunto de pasos simple y descriptivo que llamamos instrucciones. Cada instrucción crea una nueva capa en nuestra imagen. Las instrucciones incluyen acciones como:
 - Agregar un archivo o directorio
 - Ejecutar un comando
 - Crear una variable de entorno
 - Ejecutar un proceso al lanzar un contenedor
- Las capas se pueden consultar con:
 - docker image history getting-started
- Estas instrucciones se almacenan en un archivo llamado Dockerfile. Docker lee este Dockerfile cuando solicita la creación de una imagen, ejecuta las instrucciones y devuelve una imagen final.

© JMA 2020. All rights reserved

Creación de imágenes

- Hay dos formas de crear una imagen Docker:
 - A través del comando docker commit
 - A través del comando docker build con un Dockerfile (mucho mas flexible)
- Puede ser útil confirmar los cambios en ficheros o la configuración de un contenedor en una nueva imagen. Esto permite configurar y depurar un contenedor ejecutando una shell interactiva o exportar un conjunto de datos de trabajo a otro servidor, y una vez verificados los cambios confirmarlos en una nueva imagen, convirtiendo la capa contenedora en una capa mas de la imagen.
- Para inspeccionar los cambios en la capa contenedora:
 - docker diff mi-contenedor
- La operación de confirmación no incluirá ningún dato contenido en los volúmenes montados dentro del contenedor.
 - docker commit mi-contenedor testimage:version3
- Generalmente, es mejor usar Dockerfiles para administrar las imágenes de una manera documentada y mantenible.

© JMA 2020. All rights reserved

Dockerfile

- Docker puede crear imágenes automáticamente leyendo las instrucciones de un archivo Dockerfile. Un Dockerfile es un documento de texto que contiene todos los comandos que un usuario podría llamar en la línea de comando para ensamblar una imagen.
- El comando docker build crea una imagen a partir de un Dockerfile y un contexto.
- El contexto de la compilación es el conjunto de archivos en una ubicación especificada por PATH (directorio en el sistema de archivos local) o por URL (ubicación de un repositorio Git). Para aumentar el rendimiento de la compilación, se pueden excluir archivos y directorios agregando un archivo .dockerignore al directorio de contexto con los patrones de las exclusiones.
- Tradicionalmente, el fichero se denomina Dockerfile y se ubica en la raíz del contexto. Con -f se puede indicar un fichero con otro nombre en cualquier lugar del sistema de archivos. Con -t se establece la etiqueta de la imagen, un nombre legible, y se puede especificar también un repositorio.
 - docker build -f /path/to/debug.dockerfile -t jamarton/tutorial .
- El demonio Docker ejecuta las instrucciones Dockerfile una por una, asignando el resultado de cada instrucción a una nueva imagen si es necesario, antes de finalmente generar la ID de su nueva imagen. El demonio Docker limpiará automáticamente el contexto que envió.
- Siempre que sea posible, Docker utiliza una caché de compilación para acelerar el proceso docker build de manera significativa.

© JMA 2020. All rights reserved

Formato

- Las instrucciones no distinguen entre mayúsculas y minúsculas. Sin embargo, la convención es que estén en MAYÚSCULAS para distinguirlos de los argumentos más fácilmente, al comienzo de la línea (se desaconsejan espacios en blanco iniciales).
- Docker trata las líneas que comienzan con # como un comentario, a menos que la línea sea una directiva de analizador válida . Una # en cualquier otra parte de una línea se trata como un literal. Las líneas de comentario se eliminan antes de que se ejecuten las instrucciones del Dockerfile.
- Las directivas del analizador son opcionales y afectan la forma en que se manejan las líneas subsiguientes en un Dockerfile. Las directivas no agregan capas a la compilación, por lo que no se mostrarán como un paso de compilación, y se escriben como un tipo especial de comentario en el formulario # directive=value. Una directiva solo puede usarse una vez.

```
# Ejemplo del curso
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
# comando a ejecutar dentro del contenedor
CMD python /app/app.py
```

© JMA 2020. All rights reserved

FROM

- Docker ejecuta las instrucciones Dockerfile en orden. El Dockerfile debe comenzar con una instrucción FROM, aunque puede ir precedida de directivas de analizador, comentarios o instrucciones ARG que se usen en el FROM.
`FROM [-platform=<platform>] <image>[:<tag>] [AS <name>]`
- La instrucción FROM especifica la imagen base a partir de la cual se está construyendo la nueva imagen, inicializa una nueva etapa de construcción y establece la base para las instrucciones posteriores.
- La imagen puede ser cualquier imagen válida, habitualmente extraída de los repositorios públicos. Los valores tag son opcionales, el constructor asume una etiqueta latest de forma predeterminada y devuelve un error si no puede encontrar el valor tag.
- FROM puede aparecer varias veces dentro de un Dockerfile para crear varias imágenes o usar una etapa de construcción como dependencia para otra. Cada instrucción FROM borra cualquier estado creado por instrucciones anteriores. Opcionalmente, se puede dar nombre a una etapa de construcción agregando AS name a la instrucción FROM, para utilizarlo en instrucciones posteriores:
`COPY --from=<name>`
- Con --platform se puede usar para especificar la plataforma de la imagen en caso de que FROM haga referencia a una imagen multiplataforma.

© JMA 2020. All rights reserved

ARG y ENV

- Se puede usar una instrucción ARG o una ENV para especificar variables que están disponibles para la instrucción RUN y otras. Las variables de entorno definidas mediante ENV siempre anulan a los argumentos ARG con el mismo nombre.
- La instrucción ARG declara una variable argumento cuyo valor se puede pasar en el comando docker build usando --build-arg <varname>=<value> para personalizar la construcción de la imagen.
`ARG <name>[=<default value>]`
- La instrucción ENV establece una variable de entorno que persiste en la imagen y el contenedor. El valor se puede pasar cuando se ejecuta el contenedor con docker run -e <varname>=<value>. Las variables de entorno permiten personalizar tanto la construcción de la imagen como, sobre todo, la ejecución del contenedor.
`ENV <name>[=<default value>]`
- Las variables se referencian mediante la notación \${variable_name} o \${variable_name} (espacios en blanco). La sintaxis también admite algunos de los modificadores bash estándar: \${variable:-defecto} para el valor por defecto cuando el valor no está definido y \${variable:+constante} valor constante a utilizar si el valor está definido (paso de variables sin valor) o cadena vacía si no lo está.
`ARG CODE_VERSION=latest
FROM base:${CODE_VERSION}`

© JMA 2020. All rights reserved

Añadir contenido a la imagen

- La instrucción ADD copia nuevos archivos, directorios o direcciones URL de archivos remotos y los agrega al sistema de archivos de la imagen.
ADD [--chown=<user>:<group>] <src>... <dest>
- Para copiar el archivos del directorio de contexto a un determinado directorio de la imagen (en el origen acepta comodines, listas separadas por comas, ...)
ADD software.lic /opt/application/software.lic
- La fuente del archivo puede ser una URL (con / final en el src usa dest como nombre del fichero, sin / final deduce el nombre del destino del último segmento):
ADD http://wordpress.org/download/ /root/wordpress.zip
ADD http://wordpress.org/latest.zip /root/download
ADD --keep-git-dir=true https://github.com/moby/buildkit.git#v0.10.1 /buildkit
- La instrucción COPY es muy similar a ADD pero solo copia archivos locales del contexto de compilación y no tiene ninguna capacidad de extracción o descompresión.
COPY . app
COPY test.txt /absoluteDir/

© JMA 2020. All rights reserved

Comandos

- La instrucción RUN ejecutará cualquier comando en una nueva capa encima de la imagen actual y confirmará los resultados. La imagen confirmada resultante se usará para el siguiente paso en el archivo Dockerfile.
RUN <command> *modo shell*
RUN ["executable", "param1", "param2"] *modo exec*
- En modo shell, el comando se ejecuta en una shell que por defecto es `[/bin/sh", "-c"]` en Linux y `["cmd", "/S", "/C"]` en Windows.
RUN npm install --production
- El modo exec hace posible evitar la manipulación de cadenas de shell y los comandos RUN que usan una imagen base que no contiene el ejecutable de shell especificado.
RUN ["./mvnw", "dependency:resolve"]
- La instrucción SHELL permite cambiar la shell predeterminada de los comandos:
SHELL ["powershell", "-command"]
RUN Write-Host hello
Executed as powershell -command Write-Host hello

© JMA 2020. All rights reserved

Comandos

- Los comandos deben tener una sola línea salvo que se concatenen:

```
RUN useradd remote_user && \
    echo "P@$$w0rd" | passwd remote_user --stdin && \
    mkdir /home/remote_user/bin && \
    chmod 700 /home/remote_user/bin
```

- Se puede generar en modo documento un script para ejecutarlo:

```
RUN <<EOT bash
    apt-get update
    apt-get install -y nginx
EOT
```

- Se recomienda el uso de gestores de paquetes para la descarga e instalación de aplicación: yum, apt, dnf, pkg, ... (según corresponda al SO de la imagen base)
- RUN --mount permite crear montajes a los que puede acceder el proceso que se ejecuta como parte de la compilación. Esto se puede usar para vincular archivos de otra parte de la compilación sin copiar, acceder a secretos de compilación o sockets de agente ssh, o crear ubicaciones de caché para acelerar su compilación.

© JMA 2020. All rights reserved

Punto de entrada

- La instrucción ENTRYPOINT especifica el comando a ejecutar cuando se inicia un contenedor, si se desea que el contenedor ejecute el mismo ejecutable todas las veces. Es similar a la instrucción RUN, pero en lugar de ejecutar el comando cuando se está construyendo el contenedor, se especifica el comando que se ejecutará cuando se inicie el contenedor.

```
ENTRYPOINT ["java", "-jar", "/usr/app.jar"]
```

- El objetivo principal de la instrucción CMD es proporcionar valores predeterminados para un contenedor en ejecución. Estos valores predeterminados pueden incluir un ejecutable o pueden omitir el ejecutable, en cuyo caso también se debe especificar una instrucción ENTRYPOINT y se usa CMD para proporcionar sus argumentos predeterminados. Solo puede haber una instrucción CMD en un Dockerfile o solo tendrá efecto la última.

CMD ["executable","param1","param2"]	<i>modo exec (recomendado)</i>
--------------------------------------	--------------------------------

CMD command param1 param2	<i>modo shell</i>
---------------------------	-------------------

CMD ["param1","param2"]	<i>parámetros predeterminados del ENTRYPOINT</i>
-------------------------	--

CMD ["node", "server.js"]	
-----------------------------	--

- El Dockerfile debe especificar al menos uno de los comandos: CMD o ENTRYPOINT. ENTRYPOINT es el apropiado para usar el contenedor como ejecutable. Los argumentos del comando docker run <image> se agregarán después del ENTRYPOINT y anularán todos los especificados usando CMD.

© JMA 2020. All rights reserved

Personalización

- La instrucción VOLUME crea un punto de montaje con el nombre especificado y lo marca como que contiene volúmenes montados externamente desde un host nativo u otros contenedores. El valor puede ser único o múltiple. El comando docker run inicializa el volumen recién creado con cualquier dato que exista en la ubicación especificada dentro de la imagen base.

```
VOLUME /data  
VOLUME /var/log /var/db
```

- La instrucción EXPOSE informa a Docker que el contenedor escuchará en los puertos de red especificados, TCP (predeterminado) o UDP, en tiempo de ejecución. Al ejecutar el contenedor con docker run -p asignado:expuesto se deben enlazar uno a uno los puertos expuestos con los puertos asignados, solo aquellos que se quieran redirigir en el host. Funciona como un tipo de documentación para indicar los puertos que utiliza el ENTRYPPOINT.

```
EXPOSE 80  
EXPOSE 80/tcp  
EXPOSE 80/udp
```

- Docker admite la creación de redes para la comunicación entre contenedores sin necesidad de exponer o publicar puertos específicos, porque los contenedores conectados a la red pueden comunicarse entre sí a través de cualquier puerto.

© JMA 2020. All rights reserved

Entorno

- La instrucción WORKDIR establece el directorio de trabajo para las siguientes instrucciones RUN, CMD, ENTRYPPOINT, COPY y ADD. Se puede utilizar varias veces en un archivo Dockerfile pero si se proporciona una ruta relativa, será relativa a la ruta de la instrucción WORKDIR anterior. Puede utilizar variables (ARG y ENV).

WORKDIR \$DIRPATH/\$DIRNAME
- La instrucción USER establece el nombre de usuario (o UID) y, opcionalmente, el grupo de usuarios (o GID) para usar como usuario y grupo predeterminados durante el resto de la etapa actual. El usuario especificado se utiliza para las instrucciones RUN y, en tiempo de ejecución, ejecuta los comandos ENTRYPPOINT y CMD pertinentes. El usuario debe existir en el SO (root) o crearse antes de establecerlo como predeterminado. Puede cambiar tantas veces como sea necesario.

USER remote_user

© JMA 2020. All rights reserved

Documentación

- LABEL <key>=<value> <key>=<value> <key>=<value> ...
- La instrucción LABEL agrega metadatos a una imagen. Un LABEL es un par clave-valor, pero para incluir espacios hay que usar comillas y barras invertidas para múltiples líneas. Se pueden definir varios pares en la misma instrucción.

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo" version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```
- Las etiquetas incluidas en las imágenes base o principal se heredan en las nuevas imágenes. Si existe mas de una etiqueta con la misma clave pero con diferente valor, el valor más recientemente anula cualquier valor establecido anteriormente o heredado.
- La instrucción MAINTAINER (obsoleta) establece el campo Autor de las imágenes generadas, pero la instrucción LABEL es una versión mucho más flexible y debe ser usada en su lugar.

```
MAINTAINER "Javier Martín"
LABEL com.example.image.authors="jmartin@example.com"
```

© JMA 2020. All rights reserved

Construcciones de varias etapas

- Las construcciones de varias etapas permiten separar las dependencias en tiempo de compilación de las dependencias en tiempo de ejecución y reducir el tamaño general de la imagen enviando solo lo que la aplicación necesita para ejecutarse:

```
### STAGE 1: Build ####
FROM node:alpine AS build
WORKDIR /usr/src/app
COPY package.json package-lock.json .
RUN npm install
COPY .
RUN npm run build
### STAGE 2: Deploy ####
FROM nginx:alpine
COPY --from=build /usr/src/app/dist/frontend /usr/share/nginx/html
COPY ./nginx.default.conf /etc/nginx/conf.d/default.conf
#EXPOSE 80
#CMD ["nginx", "-g", "daemon off;"]
```

© JMA 2020. All rights reserved

Escaneo de seguridad

- La cadena es tan fuerte como el eslabón más débil. Las vulnerabilidades de código están aumentando en frecuencia e impacto. Dado que el software se compone cada vez más de piezas de diferentes proveedores, a las que a menudo se hace referencia como la cadena de suministro de software, puede resultar difícil encontrarlas y repararlas rápidamente. Hay que realizar un seguimiento del uso de bibliotecas y marcos, aplicaciones, contenedores, sistemas operativos, firmware, hardware y servicios.
- Cuando se haya creado una imagen, es una buena práctica escanearla en busca de vulnerabilidades de seguridad usando el comando docker scan. Docker se ha asociado con Snyk (<https://snyk.io/>) para proporcionar el servicio de análisis de vulnerabilidades.
 - docker scan tutorial
- El resultado enumera el tipo de vulnerabilidad, una URL para obtener más información y, lo que es más importante, qué versión de la biblioteca relevante corrige la vulnerabilidad.
- Se debe iniciar sesión en Docker Hub para escanear sus imágenes. Para otros proveedores:
 - docker scan --login --token SNYK_AUTH_TOKEN
- El escaneo debe realizarse periódicamente para detectar las nuevas vulnerabilidades aparecidas. En las suscripciones de pago, se puede configurar Docker Hub para escanear automáticamente todas las imágenes enviadas y luego puede ver los resultados tanto en Docker Hub como en Docker Desktop.

© JMA 2020. All rights reserved

Gestionar imágenes

- Para listar las imágenes disponibles localmente:
 - docker images
 - docker image ls
- Para buscar imágenes en Docker Hub:
 - docker search --filter is-official=true alpine
- Para mostrar información detallada sobre una o más imágenes:
 - docker image inspect tutorial-image
- Para Eliminar una o más imágenes
 - docker image rm tutorial-image
 - docker rmi tutorial-image
- Para eliminar todas las imágenes no utilizadas (sin contenedor)
 - docker image prune

© JMA 2020. All rights reserved

Distribuir imágenes

- Para compartir y distribuir imágenes existen dos mecanismos principales:
 - usar un registro de Docker (recomendado)
 - Exportar e importar en archivos tar
- Para guardar una o más imágenes en un archivo tar:
 - docker image save --output tutorial.tar af6dd155836c
 - docker image save af6dd155836c > tutorial.tar
- Para cargar una imagen desde un archivo tar o STDIN
 - docker image load --input tutorial.tar
 - docker image load < tutorial.tar

© JMA 2020. All rights reserved

Etiquetar imágenes

- Al crear una imagen, el demonio Docker genera un identificador UUID:
 - largo: “f78375b1c487e03c9438c729345e54db9d20cfac1fc3494b6eb60872e74778”
 - corto (12): “f78375b1c487”
- Definir un nombre puede ser una forma práctica de agregar significado a una imagen. Un nombre de imagen se compone de partes separadas por barras, opcionalmente con el prefijo de registro: nombre de usuario o URL del host. Las partes del nombre pueden contener letras minúsculas, dígitos y separadores (un punto, uno o dos guiones bajos o uno o más guiones) pero no puede comenzar ni terminar con un separador.
- El nombre puede incluir una etiqueta para identificar la versión (latest por defecto). El nombre de etiqueta debe ser ASCII válido y puede contener letras mayúsculas y minúsculas, dígitos, guiones bajos, puntos y guiones, no puede comenzar con un punto o un guión y puede contener un máximo de 128 caracteres.
- El nombre y etiqueta se puede asignar:
 - Al construir la imagen: docker build -t tutorial:1.0
 - Con el comando: docker tag

© JMA 2020. All rights reserved

Etiquetar imágenes

- Para asignar un nombre por comando:
 - docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
- Para asignar un nombre a través del image id
 - docker tag af6dd155836c mi-nombre:1.0.0
- Para asignar un nombre alternativo a una imagen con nombre:
 - docker tag mi-nombre otro-nombre
- Para asignar un nombre de registro en Docker Hub:
 - docker tag af6dd155836c mi-usuario/mi-repositorio:1.0.0
- Para asignar un nombre de registro en un Docker Registry propio:
 - docker tag af6dd155836c localhost:5000/mi-repositorio

© JMA 2020. All rights reserved

Registro

- Para compartir imágenes de Docker, se debe usar un registro de Docker. Las alternativas son:
 - Usar Docker Hub
 - Usar otros proveedores
 - Crear un registro propio
- El registro predeterminado es Docker Hub y es de donde provienen las imágenes oficiales usadas como base del resto de imágenes.
 - <https://hub.docker.com/>
- En Docker Hub, los usuarios disponen de repositorios públicos gratuitos para almacenar y compartir imágenes (solo 1 repositorio privado) o pueden elegir un plan de suscripción de pago para disponer de repositorios privados.
- Para enviar una imagen, primero debemos crear una cuenta en Docker Hub. Las cuentas publicas son gratuitas. Para iniciar sesión en Docker Hub desde la consola:
 - docker login -u MI-USER-NAME -p *No-recomendable*
 - cat ~/my_password.txt | docker login --username MI-USER-NAME --password-stdin

© JMA 2020. All rights reserved

Registro en Docker Hub

- Regístrese en Docker Hub para obtener una cuenta si no dispone de una.
- Para crear un repositorio privado (opcional):
 - Inicie sesión.
 - Haga clic en Crear un repositorio en la página de bienvenida de Docker Hub.
 - Asígnele el nombre mi-username/mi-repo y una descripción.
 - Establezca la visibilidad Privado.
- Para subir (empujar) una imagen al repositorio:
 - Iniciar sesión en Docker Hub en la consola:
 - docker login -u mi-username
 - Asignar a la imagen una etiqueta con el nombre del repositorio que incluye el nombre de usuario y la versión (latest por defecto):
 - docker tag getting-started mi-username/mi-repo:1.0
 - Empujar o instalar la imagen en el servidor de registro:
 - docker push mi-username/mi-repo:1.0
 - Iniciar sesión en el sitio web de Docker Hub con el navegador y completar la documentación del repositorio.
- Si no se ha creado el repositorio manualmente, al hacer el push se creara automáticamente un repositorio publico.

© JMA 2020. All rights reserved

Docker Registry

- Afortunadamente, el equipo de Docker, Inc., ha abierto el código que utilizan para ejecutar un registro de Docker, lo que nos permite construir nuestro propio registro interno.
- Docker Registry es una aplicación del lado del servidor altamente escalable y sin estado que almacena y permite distribuir imágenes de Docker. Docker Registry es de código abierto, bajo la licencia permisiva de Apache .
- Puedes utilizar Docker Registry si deseas:
 - controlar estrictamente dónde se almacenan tus imágenes
 - ser totalmente dueño del canal de distribución de tus imágenes
 - integrar el almacenamiento y la distribución de imágenes estrechamente en tu flujo de trabajo de desarrollo interno

© JMA 2020. All rights reserved

Docker Registry

- Iniciar tu registro
 - docker run -d -p 5000:5000 --name registry registry:2
- Extraer (o crear) alguna imagen del hub
 - docker pull alpine
- Etiquetar la imagen para que apunte a tu registro
 - docker image tag alpine localhost:5000/my-private-image
- Subir la imagen a tu registro
 - docker push localhost:5000/my-private-image
- Consultar las imágenes de tu registro
 - curl -X GET http://localhost:5000/v2/_catalog
- Eliminar la imagen local
 - docker image rm localhost:5000/my-private-image
- Obtener la imagen de tu registro
 - docker pull localhost:5000/my-private-image

© JMA 2020. All rights reserved

CONTENEDORES

© JMA 2020. All rights reserved

Contenedores

- Un contenedor representa la ejecución de una sola aplicación, proceso o servicio. Un contenedor es una instancia de una imagen de Docker. Se pueden crear varias instancias de un contenedor a partir de la misma imagen que son independientes entre si.
- Los contenedores aíslan el software de su entorno y aseguran que funcione de manera uniforme a pesar de las diferencias, por ejemplo, entre el desarrollo y producción.
- Las imágenes de contenedores se convierten en contenedores en tiempo de ejecución. Al crear un contenedor, el demonio docker crea una capa de grabable contenedor sobre la imagen especificada y la prepara para ejecutar el comando especificado. Al crear el contenedor se puede personalizar definiendo variables de entorno, puertos, volúmenes, ...
- Los contenedores se pueden crear y gestionar desde: Docker CLI, Docker API y Docker Dashboard.

© JMA 2020. All rights reserved

Gestionar el ciclo de vida del contenedor

- Crear un nuevo contenedor
 - docker create tutorial-image
- Iniciar uno o más contenedores detenidos
 - docker start tutorial
- Detener o matar uno o más contenedores en ejecución
 - docker stop tutorial
 - docker kill tutorial
- Reiniciar uno o más contenedores
 - docker restart tutorial
- Bloquear el proceso hasta que uno o más contenedores se detengan (docker stop) desde otro proceso, cuando se desbloque se imprimirán los códigos de salida
 - docker wait tutorial
- Eliminar uno o más contenedores (-f fuerza la detención para poder borrarlo)
 - docker rm -f tutorial
- Eliminar todos los contenedores parados
 - docker container prune

© JMA 2020. All rights reserved

Creación y ejecución

- El comando docker container create (o abreviado: docker create) crea un nuevo contenedor a partir de la imagen especificada, sin iniciararlo. Esto es útil cuando desea establecer una configuración de contenedor con anticipación para que esté listo para comenzar cuando lo necesite.
- El comando docker create comparte la mayoría de sus opciones con el comando docker run (que realiza un docker create antes de iniciararlo).
 - docker create --name tutorial -p 8080:80 -e NODE_ENV=production -v tutorial-logs:/etc/logs --restart always tutorial-image
- Los parámetros habituales con create son:
 - name tutorial Asignar un nombre al contenedor.
 - p 8080:80 Asignar el puerto 8080 del host al puerto 80 en el contenedor. (-P asigna puertos aleatorios a todos los puertos expuestos)
 - e NODE_ENV=production Asignar valores a las variables de entorno
 - v tutorial-logs:/etc/logs Enlazar volúmenes
 - restart always Reiniciar siempre si se detiene el contenedor
 - tutorial-image Especificar la imagen a utilizar.
- Los parámetros propios de run son:
 - d Ejecutar el contenedor en modo separado (en segundo plano, liberando la consola)

© JMA 2020. All rights reserved

Procesos del contenedores

- Mostrar los procesos en ejecución de un contenedor, el contenedor debe estar en ejecución
 - docker top tutorial
- Pausa todos los procesos dentro de uno o más contenedores, el contenedor debe estar en ejecución y sigue en ejecución, pero deja de atender las peticiones
 - docker pause tutorial
- Reanudar todos los procesos dentro de uno o más contenedores, el contenedor debe estar en ejecución
 - docker unpause tutorial

© JMA 2020. All rights reserved

Consultar contenedores

- Listar contenedores (en ejecución por defecto)
 - docker ls
 - docker ls -a
- Mostrar las estadísticas de uso de recursos de contenedores (transmisión en vivo)
 - docker stats tutorial
- Obtener los registros de un contenedor
 - docker logs tutorial
- Mostrar información detallada sobre uno o más contenedores
 - docker inspect tutorial
- Inspeccionar cambios en archivos o directorios en el sistema de archivos de un contenedor
 - docker diff tutorial
- Lista de asignaciones de puertos o una asignación específica para el contenedor
 - docker port tutorial

© JMA 2020. All rights reserved

Modificar contenedores

- Cambiar el nombre de un contenedor, el contenedor puede estar en ejecución
 - docker rename tutorial tutorial-new
- Actualizar la configuración de un contenedor, solo se pueden modificar los parámetros de creación referentes a la configuración de CPU, memoria y modo de restart.
 - docker update
- Vincular los flujos STDIN, STDOUT y STDERR de un contenedor en ejecución al terminal local
 - docker attach tutorial
- Copiar archivos/carpetas entre un contenedor y el sistema de archivos local
 - docker cp ./data tutorial:/app/data
 - docker cp tutorial:/var/logs/ /tmp/app_logs
- Exportar el sistema de archivos de un contenedor como un archivo tar
 - docker export --output="latest.tar" tutorial
- Crear una nueva imagen a partir de los cambios de un contenedor
 - docker commit --pause tutorial tutorial-image:2.0

© JMA 2020. All rights reserved

Ejecución dentro de un contenedor

- El comando docker exec permite ejecutar un comando en un contenedor en ejecución. El comando solo se ejecuta mientras se ejecuta el proceso principal del contenedor (PID 1), y no se reinicia si se reinicia el contenedor.
- El comando debe ser un ejecutable, un comando encadenado o entre comillas no funcionará si no se precede por sh -c, y se ejecutará en el directorio predeterminado del contenedor o en el WORKDIR si lo definió la imagen.
 - docker exec -d tutorial apt update
- Con -d se ejecuta en segundo plano. Con -it se ejecuta en modo interactivo.
 - docker exec -it tutorial sh
- Con el comando docker run se pueden ejecutar contenedores efímeros: se crean, se ejecutan y se destruyen (--rm elimina el contenedor al salir) en un único comando. Donde -i entrada y -t salida (-it) establecen el modo interactivo.
 - docker run --name ubuntu_bash --rm -i -t ubuntu bash

© JMA 2020. All rights reserved

ALMACENAMIENTO

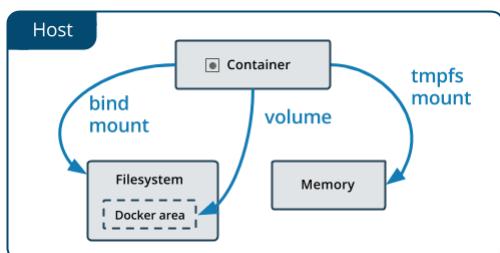
© JMA 2020. All rights reserved

Almacenamiento

- De forma predeterminada, todos los archivos creados dentro de un contenedor se almacenan en una capa de escritura del contenedor. Esto significa que:
 - Los datos no persisten cuando el contenedor se borra y puede ser difícil sacar los datos del contenedor si otro proceso los necesita.
 - La capa de escritura de un contenedor está estrechamente acoplada a la máquina host donde se ejecuta el contenedor. No se pueden mover fácilmente los datos a otro lugar.
 - Escribir en la capa de escritura de un contenedor requiere un controlador de almacenamiento para administrar el sistema de archivos que proporciona un sistema de archivos de unión, utilizando el kernel de Linux, que reduce el rendimiento en comparación con otras opciones.
- Docker tiene tres opciones para el almacenamiento de archivos, aunque desde dentro del contenedor los datos se ven siempre iguales (directorios en el sistema de archivos del contenedor):
 - Capa de escritura del contenedor: Los archivos existen mientras exista el contenedor.
 - Sistema de ficheros: Los contenedores almacenan los archivos en la máquina host, de modo que los archivos persistan incluso después de que el contenedor se detenga o elimine. Hay dos opciones: volúmenes con nombre y montajes por vinculación.
 - Memoria: Admite contenedores que almacenan archivos en memoria en la máquina host, con montaje tmpfs en Linux y npipe (canalización con nombre) en Windows. Dichos archivos desaparecen al parar el contenedor.

© JMA 2020. All rights reserved

Opciones



- **Volúmenes con nombre o volúmenes** se almacenan en una parte del sistema de archivos del host que es administrado por Docker, son la mejor manera de conservar los datos en Docker. Los procesos que no son de Docker no deben modificar esta parte del sistema de archivos.
- **Montajes por vinculación (Bind mounts)** enlazan un directorio del filesystem del host al contenedor y se pueden almacenar en cualquier parte del sistema host. Son accesibles por el contenedor, otros contenedores y cualquier proceso del host, que pueden modificarlos en cualquier momento.
- **tmpfs y npipe** se almacenan únicamente en la memoria del sistema host y nunca se escriben en el sistema de archivos del sistema host. Solo son accesibles por el contenedor.

© JMA 2020. All rights reserved

Casos de uso de los volúmenes

- Compartir datos entre varios contenedores en ejecución. Si no se crea explícitamente, se crea un volumen la primera vez que se monta en un contenedor. Cuando ese contenedor se detiene o se retira, el volumen aún existe, solo pueden eliminar explícitamente. Varios contenedores pueden montar el mismo volumen simultáneamente, ya sea de lectura y escritura o de solo lectura.
- Cuando no se garantiza que el host de Docker tenga un directorio o una estructura de archivos determinados. Los volúmenes ayudan a desacoplar la configuración del host de Docker del contenedor en tiempo de ejecución.
- Cuando se desee almacenar los datos del contenedor en un host remoto o en un proveedor de la nube, en lugar de localmente.
- Cuando se necesite realizar una copia de seguridad, restaurar o migrar datos de un host Docker a otro, los volúmenes son una mejor opción. Se pueden detener los contenedores que usan el volumen y luego hacer una copia de seguridad del directorio del volumen en el host.
- Cuando la aplicación del contenedor requiere E/S de alto rendimiento en Docker Desktop. Los volúmenes se almacenan en la máquina virtual de Linux en lugar del host, lo que significa que las lecturas y escrituras tienen una latencia mucho más baja y un rendimiento más alto.
- Cuando la aplicación del contenedor requiere un comportamiento completamente nativo del sistema de archivos, como un motor de base de datos que requiere un control preciso sobre el vaciado del disco para garantizar la durabilidad de las transacciones. Los volúmenes se almacenan en la máquina virtual de Linux y pueden dar estas garantías, mientras que los montajes por vinculación se envían de forma remota a macOS o Windows, donde los sistemas de archivos se comportan de manera ligeramente diferente.

© JMA 2020. All rights reserved

Casos de uso de los montajes por vinculación

- Compartir archivos de configuración desde la máquina host a los contenedores. Así es como Docker proporciona la resolución de DNS a los contenedores de forma predeterminada, montándolos /etc/resolv.conf desde la máquina host en cada contenedor.
- Compartir código fuente o crear artefactos entre un entorno de desarrollo en el host de Docker y un contenedor. Por ejemplo, puede montar un directorio target/de Maven en un contenedor y, cada vez que se compila el proyecto Maven en el host de Docker, el contenedor obtiene acceso a los artefactos reconstruidos. Si se usa de esta manera, el Dockerfile de producción debería copiar los artefactos listos para producción directamente en la imagen, en lugar de depender de un montaje por vinculación.
- Cuando se garantiza que la estructura de archivos o directorios del host de Docker es coherente con los montajes por vinculación que requieren los contenedores.

© JMA 2020. All rights reserved

Volúmenes

- Docker crea y administra completamente los volúmenes con nombre y están aislados de la funcionalidad principal de la máquina host. Los volúmenes son el mecanismo preferido para conservar los datos generados y utilizados por los contenedores de Docker. Los volúmenes tienen varias ventajas sobre los montajes por vinculación:
 - Los volúmenes son más fáciles de respaldar o migrar que los montajes enlazados.
 - Puede administrar volúmenes mediante los comandos de la CLI de Docker o la API de Docker.
 - Los volúmenes funcionan en contenedores de Linux y Windows.
 - Los volúmenes se pueden compartir de forma más segura entre varios contenedores.
 - Los volúmenes admiten el uso *controladores de volumen* que permiten almacenar volúmenes en hosts remotos o proveedores de la nube, cifrar el contenido de los volúmenes o agregar otras funcionalidades.
 - Los nuevos volúmenes pueden tener su contenido rellenado previamente por un contenedor.
 - Los volúmenes en Docker Desktop tienen un rendimiento mucho mayor que los montajes por vinculación de los hosts de Mac y Windows.
- Un volumen no aumenta el tamaño de los contenedores que lo utilizan y el contenido del volumen existe fuera del ciclo de vida de un contenedor determinado.

© JMA 2020. All rights reserved

Crear volúmenes

- Los volúmenes se pueden crear explícitamente o dejar que se creen automáticamente cuando se montan en un contenedor.
- Cuando se crea un volumen, se almacena dentro de un directorio en el host de Docker (/var/lib/docker/volumes/ en Linux). Cuando se monta el volumen en un contenedor, este directorio es lo que se monta en el contenedor.
 - docker volume create mi-volumen
- Un volumen puede tener un nombre o ser anónimo. Los volúmenes anónimos no reciben un nombre explícito cuando se crean o se montan por primera vez en un contenedor, por lo que Docker les otorga un nombre aleatorio (hash) que garantiza que es único dentro de un host de Docker determinado. Los volúmenes con nombre son más cómodos de utilizar, es la única diferencia con los anónimos.
- Un volumen dado se puede montar en varios contenedores simultáneamente. Cuando ningún contenedor en ejecución usa un volumen, el volumen aún está disponible para Docker y no se elimina automáticamente.

© JMA 2020. All rights reserved

Administrar volúmenes

- Para listar todos los volúmenes disponibles:
 - docker volume ls
- Para inspeccionar un volumen:
 - docker volume inspect mi-volumen
- Para eliminar un volumen concreto, no se pueden eliminar volúmenes en uso aunque se intente forzar con -f, podría dejar inestables los contenedores que lo utilizan:
 - docker volume rm mi-volumen
- Para eliminar todos los volúmenes que no estén uso:
 - docker volume prune

© JMA 2020. All rights reserved

Montar volúmenes en contenedores

- Los volúmenes se montan en la creación del contenedor con los parámetros -v o --volumen, sintaxis combinada, o con --mount, sintaxis separada. Un contenedor puede montar tantos volúmenes como sean necesarios.
 - El parámetro -v consta de tres campos, separados por dos puntos (:) que deben estar en el orden correcto (por lo que el significado de cada campo no es inmediatamente obvio).
 - En el caso de volúmenes con nombre, el primer campo es el nombre del volumen y es único en una máquina host determinada. Para volúmenes anónimos, se omite el primer campo.
 - El segundo campo es la ruta donde se monta el archivo o directorio dentro del contenedor.
 - El tercer campo es opcional y es una lista de opciones de montaje separadas por comas, como ro (readonly).
- docker run -v mi-volumen:/app/data:ro -v /ruta/en/contenedor -d ...

© JMA 2020. All rights reserved

Montar volúmenes en contenedores

- El parámetro `--mount` consta de varios pares clave-valor, separados por comas y cada uno de ellos consta de una tupla `<key>=<value>`. Esta sintaxis es más detallada y el orden no es significativo por lo que es más fácil de entender:
 - `type`: tipo de montaje que puede ser `bind`, `volume` (por defecto), o `tmpfs`.
 - `source` o `src`: para volúmenes con nombre, este es el nombre del volumen. Para volúmenes anónimos, este campo se omite.
 - `destination`, `dst` o `target` : el valor con la ruta donde se está montado el archivo o directorio en el contenedor.
 - `readonly` o `ro`: si está presente, hace que el montaje de vinculación se monte en el contenedor como de solo lectura.
 - `volume-opt`: que se puede especificar más de una vez, toma un par clave-valor con el nombre de una opción y su valor.

```
docker run --mount source=mi-volumen,target=/app/data,readonly --mount target=/ruta/en/contenedor -d ...
```

© JMA 2020. All rights reserved

Montajes por vinculación

- Los montajes por vinculación tienen una funcionalidad limitada en comparación con los volúmenes. Cuando se utiliza un montaje de vinculación, un archivo o directorio en la máquina host se monta y vincula en un contenedor. Se hace referencia al archivo o directorio por su ruta absoluta en la máquina host.
- No es necesario que el archivo o directorio ya exista en el host de Docker. Se crea bajo demanda si aún no existe. Los montajes por vinculación son muy eficaces, pero dependen de que el sistema de archivos de la máquina host tenga una estructura de directorio específica disponible. No puede usar los comandos de la CLI de Docker para administrar directamente los montajes por vinculación.
- Al montar los montajes por vinculación en los contenedores con el parámetro `-v`, el primer campo es la ruta absoluta al archivo o directorio en la máquina host. Con `--mount` el `type` debe ser `bind` y `source` la ruta absoluta al archivo o directorio en la máquina host. Las rutas absolutas empiezan por `/` para evitar confusión con los volúmenes. Se puede usar `$(pwd)` que se expande al directorio de trabajo actual en Linux o macOS, `%cd%` en Windows cmd y `$pwd` en Windows PowerShell.
 - `docker run --mount type=bind,src=$(pwd)/data,target=/app/data -v $(pwd)/logs:/logs -d ...`

© JMA 2020. All rights reserved

Inicialización del contenido

- Si monta el contenedor con un volumen vacío, o un volumen que aún no existe, apuntado a archivos o directorios que existen en la imagen del contenedor, estos archivos o directorios se copian (propagan) al volumen.
- Si monta el contenedor con un volumen no vacío se respeta el contenido del volumen, no se copia nada, independientemente del contenido en la imagen.
- A diferencia de los volúmenes, si se realiza un montaje por vinculación nunca se copia nada, aunque el directorio fuese creado por el montaje al no existir previamente e independientemente del contenido en la imagen y del directorio en el host.
- El directorio de la imagen del contenedor en el que existen algunos archivos o directorios quedan ocultos por el montaje, independientemente de la copia. Los archivos ocultos no se eliminan ni modifican, siguen en la capa de la imagen, pero no se puede acceder a ellos mientras este montado el volumen o el montaje de vinculación.

© JMA 2020. All rights reserved

Montajes en memoria

- Los montajes en memoria son apropiados para los casos en los que no desea que los datos persistan en la máquina host o dentro del contenedor. Esto puede deberse a motivos de seguridad o para proteger el rendimiento del contenedor cuando su aplicación necesita escribir un gran volumen de datos de estado no persistentes.
- Si su contenedor genera datos de estado no persistentes, usar un montaje en memoria evita almacenar los datos en cualquier lugar de forma permanente y aumenta el rendimiento del contenedor al evitar escribir en la capa de escritura del contenedor.
- Dado que el sistema de archivos es uno de los recursos mas lentos del host, los montajes en memoria, cuando se dispone de suficiente memoria, son ideales para contenedores efímeros: se crean, se ejecutan utilizando la memoria como sistema de archivos y se destruyen liberando los recursos.
- A diferencia de los volúmenes y los montajes por vinculación, no se pueden compartir los montajes en memoria entre contenedores.
- Los montajes en memoria se establecen en el contenedor con el parámetro `--tmpfs` o con `--mount` pero no puede tener `source`.
 - `docker run --mount type=tmpfs,destination=/app/data --tmpfs /logs -d ...`
- En los montajes en memoria no se inicializa el contenido.

© JMA 2020. All rights reserved

REDES

© JMA 2020. All rights reserved

Introducción

- Una de las razones principales del enfoque de contenedores es permitir el despliegue de sistemas distribuidos y su escalabilidad a múltiples instancias, lo que requiere las conexiones entre ellos.
- Docker permite a los contenedores y servicios conectarse entre sí o conectarse a cargas de trabajo que no sean de Docker. Los contenedores y servicios de Docker ni siquiera necesitan saber que están implementados en Docker, o si sus pares también son cargas de trabajo de Docker o no. Ya sea que los hosts de Docker ejecuten Linux, Windows o una combinación de ambos, se puede usar Docker para administrarlos de forma independiente de la plataforma.
- Docker dispone un subsistema de red, que es conectable y utiliza controladores, y permite crear, administrar y usar redes en Docker. Existen varios controladores de forma predeterminada que proporcionan el núcleo de la funcionalidad de red de Docker. Se pueden instalar y usar complementos de red de terceros que amplíen la funcionalidad de Docker.

© JMA 2020. All rights reserved

Controladores

- **bridge:** Las redes puente se usan cuando las aplicaciones que se ejecutan en contenedores independientes, dentro del mismo host de Docker, necesitan comunicarse entre sí. Es el controlador de red predeterminado y por defecto.
- **host:** Elimina el aislamiento de red entre los contenedores independientes y el host de Docker, todos en la misma dirección, pero permite aislar otros aspectos del contenedor.
- **overlay:** Las redes superpuestas conectan varios demonios Docker y permiten que los servicios swarm se comuniquen entre sí. También puede usar redes superpuestas para facilitar la comunicación entre un servicio swarm y un contenedor independiente, o entre dos contenedores independientes en diferentes demonios Docker. Esta estrategia elimina la necesidad de realizar enrutamiento a nivel de sistema operativo entre estos contenedores.
- **ipvlan:** Las redes IPVlan brindan a los usuarios un control total sobre las direcciones IPv4 e IPv6. El controlador VLAN se basa en eso para brindar a los operadores un control completo del etiquetado de capa 2 VLAN e incluso el enrutamiento IPVlan L3 para los usuarios interesados en la integración de la red subyacente.
- **macvlan:** Las redes Macvlan permiten asignar una dirección MAC a un contenedor, haciéndolo aparecer como un dispositivo físico en su red. El demonio Docker enruta el tráfico a los contenedores por sus direcciones MAC. Es la mejor opción cuando se trata de aplicaciones heredadas que esperan conectarse directamente a la red física, en lugar de enrutarlas a través de la pila de red del host Docker.
- **none:** Deshabilita los servicios de red para un contenedor.

© JMA 2020. All rights reserved

Bridge

- En términos de redes, una red puente es un dispositivo de la capa de enlace que reenvía el tráfico entre los segmentos de la red. Un puente puede ser un dispositivo de hardware o un dispositivo de software que se ejecuta dentro del kernel de una máquina host.
- En términos de Docker, una red puente utiliza un puente de software que permite que los contenedores conectados a la misma red puente se comuniquen entre sí, mientras proporciona aislamiento de los contenedores que no están conectados a dicha red de puente. El controlador de puente de Docker instala reglas automáticamente en la máquina host para que los contenedores en diferentes redes puente no puedan comunicarse directamente entre sí.
- Las redes puente se aplican a los contenedores que se ejecutan en el mismo host del demonio de Docker. Para la comunicación entre contenedores que se ejecutan en diferentes hosts de Docker, se puede administrar el enrutamiento a nivel del sistema operativo o puede usar una red superpuesta.
- Cuando inicia Docker, se crea automáticamente una red de puente predeterminada (llamada bridge) y los contenedores recién iniciados se conectan por defecto a ella, a menos que se especifique lo contrario. Se pueden y deben crear redes puente personalizadas.

© JMA 2020. All rights reserved

Ventajas de las redes puente personalizadas

- Las redes puente personalizadas proporcionan una resolución de DNS automática entre contenedores: los contenedores pueden resolverse entre sí por nombre o alias. En la red bridge (predeterminada), los contenedores solo pueden acceder entre sí mediante direcciones IP, a menos que se creen enlaces manualmente entre los contenedores (usando --link, obsoleto). Estos enlaces deben crearse en ambas direcciones, esto se vuelve complejo cuando más de dos contenedores deban comunicarse.
- Las redes puente personalizadas proporcionan un mejor aislamiento, solo los contenedores explícitamente conectados a esa red pueden comunicarse entre si. Todos los contenedores que no especifiquen --network se adjuntan a la red bridge, siendo un riesgo ya que permite que contenedores no relacionados puedan comunicarse.
- Las redes puente personalizadas facilitan definir diferentes requisitos de red para diferentes grupos de aplicaciones, un contenedor puede pertenecer a varias redes. La red bridge se puede configurar pero afecta a todos los contenedores que la usan por defecto, la configuración de la red de puente predeterminada reside fuera de Docker y requiere reiniciar Docker.
- Los contenedores vinculados en la red bridge comparten variables de entorno .

© JMA 2020. All rights reserved

Redes superpuestas

- El controlador de red overlay crea una red distribuida entre varios hosts de daemon de Docker. Esta red se encuentra sobre (superpone) las redes específicas de los host, lo que permite que los contenedores conectados a ella (incluidos los contenedores de servicios swarm) se comuniquen de forma segura cuando el cifrado está habilitado. Docker maneja de manera transparente el enrutamiento de cada paquete hacia y desde el host del demonio Docker correcto y el contenedor de destino correcto.
- Se pueden crear redes overlay de la misma manera que las redes puente personalizadas, utilizando el parámetro --driver overlay. Los servicios o contenedores se pueden conectar a más de una red a la vez. Los servicios o contenedores solo pueden comunicarse a través de las redes a las que están conectados.

© JMA 2020. All rights reserved

Modo host

- Cuando se usa el modo host de red (controlador de red host) para un contenedor, la pila de red de ese contenedor no está aislada del host de Docker (el contenedor comparte el espacio de nombres de red del host) y el contenedor no obtiene su propia dirección IP (utiliza la del host).
 - docker run --rm -d --network host --name web-server nginx
- Dado que el contenedor no tiene su propia dirección IP cuando se utiliza el modo host de red, el mapeo de puertos no tiene efecto y las opciones -p, --publish, -P y --publish-all se ignoran, produciendo una advertencia en su lugar, los puertos se mapean a nivel de host con la IP del host.
- La red en modo host puede ser útil para optimizar el rendimiento y en situaciones en las que un contenedor necesita manejar una gran variedad de puertos, ya que no requiere traducción de direcciones de red (NAT) y no se crea un "proxy de usuario" para cada puerto.
- El controlador de red del host solo funciona en hosts Linux y no es compatible con Docker Desktop para Mac, Docker Desktop para Windows o Docker EE para Windows Server.
- La red host se crea en el proceso de instalación de Docker Desktop.

© JMA 2020. All rights reserved

Administrar redes

- Para crear una red puente personalizadas o superpuesta (se puede especificar con parámetros la subred, el rango de direcciones IP, la puerta de enlace y otras opciones):
 - docker network create tutorial-net
- Para listar todas las redes disponibles:
 - docker network ls
- Para inspeccionar una red (no muestra los contenedores parados):
 - docker network inspect tutorial-net
- Para conectar un contenedor a una red:
 - docker network connect tutorial-net tutorial
 - docker network disconnect -f tutorial-net tutorial
- Para eliminar una red concreta, no puede tener contenedores conectados (en ejecución):
 - docker network rm tutorial-net
- Para eliminar todas las redes que no estén en uso:
 - docker network prune

© JMA 2020. All rights reserved

DOCKER COMPOSE

© JMA 2020. All rights reserved

Introducción

- Docker Compose es una herramienta que se desarrolló para ayudar a definir y compartir aplicaciones de varios contenedores. Con Compose, podemos crear un archivo YAML para definir y configurar los servicios y, con un solo comando, podemos activar o desactivar todo. Compose funciona en todos los entornos: producción, pre producción, desarrollo, pruebas, así como en flujos de trabajo de CI.
- El uso de Compose es básicamente un proceso de tres pasos:
 - Definir las imágenes de la aplicación en sus correspondientes Dockerfile para poder crear sus contenedores.
 - Definir y configurar en el fichero docker-compose.yml los servicios que componen la aplicación para que puedan ejecutarse juntos en un entorno aislado.
 - Ejecutar docker compose up que inicia y ejecuta toda la aplicación.
- Compose tiene comandos para administrar todo el ciclo de vida de su aplicación:
 - Iniciar, detener y reconstruir servicios
 - Ver el estado de los servicios en ejecución
 - Transmite la salida de registro de los servicios en ejecución
 - Ejecutar un comando único en un servicio
- El nuevo Compose V2 admite el comando compose como parte del Docker CLI, ya no requiere el docker-compose. La instalación de Docker Desktop deja instalado Docker Compose.

© JMA 2020. All rights reserved

Características

- **Múltiples entornos aislados en un solo host:** Compose usa un nombre de proyecto para aislar los entornos entre sí. El nombre predeterminado es el nombre del directorio base del proyecto (del archivo Compose). Se puede establecer el nombre mediante la variable de entorno COMPOSE_PROJECT_NAME o la opción -p de línea de comandos, así como el directorio base con la opción --project-directory. Se puede hacer uso de este nombre de proyecto en varios contextos diferentes:
 - en un host de desarrollo, para crear varias copias de un solo entorno, como cuando se desea ejecutar una copia estable para cada rama de un proyecto
 - en un servidor CI, para evitar que las compilaciones interfieran entre sí, se puede establecer el nombre del proyecto en un número de compilación único
 - en un host compartido o dev host, para evitar que diferentes proyectos, con los mismos nombres de servicio, interfieran entre sí
- **Preservar los datos de volumen cuando se crean contenedores:** Compose conserva todos los volúmenes utilizados por sus servicios. Cuando se ejecuta docker compose up, si encuentra contenedores de ejecuciones anteriores, copia los volúmenes del contenedor anterior al contenedor nuevo. Este proceso garantiza que no se pierda ningún dato que haya creado en los volúmenes.
- **Recrear solo contenedores que hayan cambiado:** Compose almacena en caché la configuración utilizada para crear un contenedor. Cuando reinicia un servicio que no ha cambiado, Compose reutiliza los contenedores existentes, permitiendo realizar cambios en el entorno muy rápidamente.
- **Variables entre entornos:** Compose admite variables en el archivo Compose. Se pueden usar estas variables para personalizar la composición para diferentes entornos o diferentes usuarios y se puede ampliar un archivo de Compose utilizando el campo extends o creando varios archivos de Compose.

© JMA 2020. All rights reserved

Casos de uso

- **En entornos de desarrollo**
 - Cuando desarrolla software, la capacidad de ejecutar una aplicación en un entorno aislado e interactuar con ella es crucial. Se puede utilizar el comando compose para crear el entorno e interactuar con él.
 - El archivo Compose proporciona una forma de documentar y configurar todas las dependencias del servicio de la aplicación (bases de datos, colas, cachés, API de servicios web, etc.). Compose permite crear e iniciar uno o más contenedores para cada dependencia con un solo comando.
 - Juntas, estas características brindan una manera conveniente para que los desarrolladores comiencen un proyecto. Compose puede reducir una "guía de inicio para desarrolladores" de varias páginas a un solo archivo de Compose legible por máquina y algunos comandos.
- **En entornos de prueba automatizados**
 - Una parte importante de cualquier proceso de implementación continua o integración continua es el conjunto de pruebas automatizado. Las pruebas automatizadas de extremo a extremo requieren un entorno limpio en el que ejecutar las pruebas. Compose proporciona una forma conveniente de crear y destruir entornos de prueba aislados para su conjunto de pruebas. Se pueden crear y destruir estos entornos con solo unos pocos comandos:

```
docker compose up -d
./run_tests
docker compose down
```

© JMA 2020. All rights reserved

Archivo Compose

- La especificación Compose permite definir una aplicación basada en contenedor independiente de la plataforma. Una aplicación de este tipo está diseñada como un conjunto de contenedores que deben funcionar juntos con recursos compartidos y canales de comunicación adecuados.
- El archivo Compose es un fichero YAML que define la versión (obsoleta), los servicios (obligatorios), las redes, los volúmenes, las configuraciones y los secretos para una aplicación Docker.
- El nombre y la ruta predeterminada para un archivo Compose es `compose.yaml` (preferido) o `compose.yml` en el directorio de trabajo. También se deberían admitir los nombres `docker-compose.yaml` y `docker-compose.yml` por la compatibilidad con versiones anteriores. Si existen ambos archivos, se elige `compose.yaml`.
- Se pueden combinar varios archivos de Compose para definir el modelo de aplicación.

© JMA 2020. All rights reserved

Modelo de aplicación

- Los componentes informáticos de una aplicación se definen como Servicios . Un Servicio es un concepto abstracto implementado en plataformas mediante la ejecución de la misma imagen de contenedor (y configuración) una o más veces.
- Los servicios se comunican entre sí a través de Redes. Una red es una abstracción de capacidad de plataforma para establecer una ruta IP entre contenedores dentro de servicios conectados entre sí. Las opciones de redes específicas de plataforma de bajo nivel se agrupan en la definición de red y pueden implementarse parcialmente en algunas plataformas.
- Los servicios almacenan y comparten datos persistentes en Volúmenes. La especificación describe datos persistentes como un montaje de sistema de archivos de alto nivel con opciones globales. Los detalles de implementación específicos de la plataforma real se agrupan en la definición de Volúmenes y pueden implementarse parcialmente en algunas plataformas.
- Algunos servicios requieren datos de Configuración que dependen del tiempo de ejecución o la plataforma. Desde el punto de vista del contenedor de servicios, las configuraciones son comparables a los volúmenes, ya que son archivos montados en el contenedor. Pero la definición real implica servicios y recursos de plataforma distintos, que se abstraen de este tipo.
- Un Secreto es un tipo específico de datos de configuración para datos confidenciales que no deben exponerse sin consideraciones de seguridad y están disponibles como archivos montados en sus contenedores.
- La distinción entre Volúmenes, Configuraciones y Secreto permite una abstracción comparable a nivel de servicio, pero que cubren la configuración específica de recursos de plataforma.

© JMA 2020. All rights reserved

Proyectos

- Un proyecto es una implementación individual de una especificación de aplicación en una plataforma. El nombre de un proyecto se utiliza para agrupar recursos y aislarlos de otras aplicaciones u otras instalaciones de la misma aplicación especificada de Compose con parámetros distintos.
- Una implementación de Compose que crea recursos en una plataforma debe prefijar los nombres de los recursos por el nombre del proyecto y establecer la etiqueta com.docker.compose.project.
- El nombre predeterminado del proyecto es el nombre base del directorio del proyecto. Puede establecer un nombre de proyecto personalizado mediante la opción -p de línea de comandos, la variable de entorno COMPOSE_PROJECT_NAME o un atributo de nivel superior name.
- El directorio del proyecto predeterminado es el directorio base del archivo Compose. Se puede definir un valor personalizado con la opción de línea de comandos --project-directory.

© JMA 2020. All rights reserved

Servicios

- Un Servicio es una definición abstracta de un recurso informático dentro de una aplicación que se puede escalar/reemplazar independientemente de otros componentes. Los servicios están respaldados por un conjunto de contenedores, ejecutados por la plataforma de acuerdo con los requisitos de replicación y las restricciones de ubicación. Al estar respaldados por contenedores, los servicios se definen mediante una imagen de Docker y un conjunto de argumentos de tiempo de ejecución. Todos los contenedores dentro de un servicio se crean de forma idéntica con estos argumentos.
- Un archivo Compose DEBE declarar un elemento raíz services con los nombres de servicios y sus definiciones de servicios.
- Una definición de servicio contiene la configuración para crear cada contenedor iniciado para ese servicio (similar al docker create) y debe incluir el elemento image que indique la imagen del contenedor o una sección build de compilación, que defina cómo crear la imagen de Docker para el servicio.

© JMA 2020. All rights reserved

Elementos habituales en los servicios

- **environment:** define las variables de entorno establecidas en el contenedor.
- **ports:** publica los puertos del contenedor
- **networks:** define las redes a las que están conectados los contenedores del servicio
- **volumes:** define rutas de host de montaje por vinculación o volúmenes con nombre que deben ser accesibles para los contenedores del servicio.
- **tmpfs:** monta un sistema de archivos en memoria dentro del contenedor.
- **entrypoint:** anula el punto de entrada predeterminado para la imagen de Docker.
- **command:** anula el comando predeterminado declarado por la imagen del contenedor (CMD).
- **restart:** define la política que la plataforma aplicará al pararse el contenedores.
- **depends_on:** expresa dependencias entre servicios que deben iniciarse o cerrarse en orden.
- **configs:** lista de las configuraciones utilizadas por el servicio.
- **secrets:** otorga acceso a datos confidenciales definidos por secretos por servicio.
- **profiles:** lista de perfiles en el que el servicio debe estar habilitado, por defecto en todos.

© JMA 2020. All rights reserved

Definición de servicio

```
services:  
  db:  
    image: mysql:5.7  
    ports:  
      - 3306:3306  
    volumes:  
      - todo-mysql-data:/var/lib/mysql  
    environment:  
      MYSQL_ROOT_PASSWORD: root  
      MYSQL_DATABASE: todos  
  networks:  
    - todo-net  
  configs:  
    - my_config  
  
  app:  
    build: .  
    ports:  
      - 3000:3000  
    environment:  
      MYSQL_HOST: db  
      MYSQL_USER: root  
      MYSQL_PASSWORD: root  
      MYSQL_DB: todos  
    command: [ "/bin/sh", "-c", "app", "-p", "3000" ]  
    networks:  
      - todo-net  
    depends_on:  
      - db
```

© JMA 2020. All rights reserved

Controlar el orden de inicio y apagado

- Con múltiples contenedores suele ser importante el orden de inicio y apagado de los mismos. El elemento depends_on de los servicios expresa las dependencias de inicio y cierre entre servicios.
- Compose debe garantizar que los servicios de dependencia se hayan iniciado antes de iniciar un servicio dependiente.
- Se puede especificar la condición bajo la cual se considera satisfecha la dependencia:
 - service_started (por defecto): espera solo a que se hayan iniciado los servicios de dependencia, y no a que estén listos, antes de iniciar un servicio dependiente.
 - service_healthy: especifica que se espera que una dependencia sea "saludable" (como lo indica el control de estado) antes de iniciar un servicio dependiente.
 - service_completed_successfully: especifica que se espera que una dependencia se ejecute correctamente antes de iniciar un servicio dependiente.

© JMA 2020. All rights reserved

Controlar el orden de inicio y apagado

```
services:  
  web:  
    build:  
      context: frontend  
      dockerfile: ./frontend.Dockerfile  
    depends_on:  
      db:  
        condition: service_healthy  
      redis:  
        condition: service_started  
    redis:  
      image: redis  
    db:  
      image: postgres
```

© JMA 2020. All rights reserved

Uso de perfiles

- Los perfiles permiten ajustar el modelo de aplicación Compose para varios usos y entornos mediante la habilitación selectiva de servicios. Esto se logra asignando cada servicio a cero o más perfiles. Si no está asignado, el servicio siempre se inicia, pero si está asignado, solo se inicia si el perfil está activado.
- Esto permite definir servicios adicionales en un solo archivo docker-compose.yml que solo deben iniciarse en escenarios específicos, por ejemplo, para tareas de depuración o pruebas.

db:

```
image: mysql
phpmyadmin:
  image: phpmyadmin
  profiles: ["debug"]
  depends_on: [ db ]
```

– docker compose --profile debug up

© JMA 2020. All rights reserved

Volúmenes

- La sección volumes de los servicios define las rutas de host de montaje por vinculación o volúmenes con nombre que deben ser accesibles para los contenedores del servicio. Permite tanto la sintaxis --volumen como la de --mount.
- La sección general volumes permite la configuración de volúmenes con nombre que se pueden reutilizar en varios servicios.
- Las entradas en sección volumes permiten configurar los diferentes volúmenes. Si esta vacía se usa la configuración predeterminada de la plataforma para crear un volumen. Con external: true se especifica que el volumen ya existe en la plataforma y su ciclo de vida se administra fuera del de la aplicación.

© JMA 2020. All rights reserved

Volúmenes

```
services:  
  backend:  
    image: mi-database  
    volumes:  
      - db-data:/etc/data  
      - db-log:/etc/logs  
      - type: bind  
        source: /backup/postgres/  
        target: /var/run/postgres/backup  
  monitoring:  
    image: my-monitoring  
    volumes:  
      - db-data:/etc/data  
  
volumes:  
  db-data:  
    driver: flocker  
    driver_opts:  
      size: "10GiB"
```

© JMA 2020. All rights reserved

Variables de entorno

- Al igual que en Dockerfile y con la misma notación \${}, se pueden utilizar variables para completar valores dentro de un archivo Compose.

```
services:  
  foo:  
    image: "webapp:${TAG}"  
    environment:  
      - COMPOSE_PROJECT_NAME  
    command: echo "I'm running ${COMPOSE_PROJECT_NAME}"
```

- Compose utiliza los valores de las variables del entorno de la shell o los indicados con las opciones de línea de comandos -e y --env-file. Se pueden establecer los valores predeterminados para cualquier variable de entorno en un archivo denominado .env en el directorio base del proyecto, docker compose up los leerá automáticamente.
- Las siguientes reglas de sintaxis se aplican al archivo .env:
 - Una línea por variable con formato: VARIABLE=VALOR
 - Las líneas que comienzan con # se procesan como comentarios y se ignoran, igual que las líneas en blanco.
 - No hay un manejo especial de las comillas. Esto significa que son parte del valor.

© JMA 2020. All rights reserved

Redes

- De forma predeterminada, Compose configura una sola red para la aplicación. Los contenedores de los servicios se unen a la red predeterminada y el resto contenedores en esa red pueden acceder a ellos y pueden descubrirlo en un nombre de host idéntico al nombre del servicio.
- La red recibe un nombre basado en el nombre del proyecto, que se basa en el nombre del directorio en el que se encuentra, con el sufijo _default.
- Si se realiza un cambio de configuración en un servicio y ejecuta docker compose up para actualizarlo, el contenedor anterior se elimina y el nuevo se une a la red con una dirección IP diferente pero con el mismo nombre. Los contenedores en ejecución pueden buscar ese nombre y conectarse a la nueva dirección, pero la dirección anterior deja de funcionar. Si algún contenedor tiene conexiones abiertas con el contenedor anterior, se cierran. Es responsabilidad del contenedor detectar esta condición, buscar el nombre nuevamente y volver a conectarse.
- Se pueden especificar redes propias con la sección networks. Esto permite crear topologías más complejas y especificar opciones y controladores de red personalizados. También se puede usar para conectar servicios a redes creadas externamente que no son administradas por Compose (definidas con external: true). Cada servicio puede especificar a qué redes conectarse a través del elemento networks, que es una lista de nombres de la sección networks. Se cambiar la configuración de la red predeterminada definiendo una entrada llamada default en la sección networks.

© JMA 2020. All rights reserved

Redes

```
services:  
  frontend:  
    image: my-webapp  
    networks:  
      - front-tier  
      - back-tier  
  monitoring:  
    image: my-monitoring  
    networks:  
      - admin  
  backend:  
    image: my-backend  
    networks:  
  
  back-tier:  
    aliases:  
      - database  
    admin:  
      aliases:  
        - mysql  
  
  networks:  
    front-tier:  
      name: my-pre-existing-network  
      external: true  
    back-tier:  
      admin:
```

© JMA 2020. All rights reserved

Configuraciones

- Las configuraciones permiten que los servicios adapten su comportamiento sin necesidad de reconstruir una imagen de Docker. Las configuraciones son comparables a los volúmenes desde el punto de vista del servicio, ya que se montan en el sistema de archivos de contenedores del servicio.
- Los servicios solo pueden acceder a las configuraciones cuando se otorgan explícitamente en su elemento configs, en cuyo caso el contenido de la configuración se monta como un archivo en el contenedor (/<config-name> en contenedores de Linux y C:\<config-name> en contenedores de Windows).
- La sección configs define o hace referencia a los datos de configuración que se pueden otorgar a varios servicios de la aplicación.

```
services:  
  redis:  
    image: redis:latest  
    configs:  
      - my_config  
configs:  
  my_config:  
    file: ./my_config.txt
```

© JMA 2020. All rights reserved

Secretos

- Los secretos son una variante de las configuraciones que se centran en datos confidenciales, con restricciones específicas para este uso, que permite la separación de responsabilidades.
- Los servicios solo pueden acceder a las configuraciones cuando se otorgan explícitamente en su elemento secrets, en cuyo caso lo monta como de solo lectura /run/secrets/<secret_name> dentro del contenedor. El nombre de origen y el punto de montaje de destino se pueden establecer al definir el secreto.
- La sección secrets define o hace referencia a los datos confidenciales que se pueden otorgar a varios servicios de la aplicación.

```
services:  
  frontend:  
    image: mi-webapp  
    secrets:  
      - server-certificate  
secrets:  
  server-certificate:  
    file: ./server.cert
```

© JMA 2020. All rights reserved

docker compose

- El comando docker compose permite acceder a los comandos específicos de Compose.
- Las principales opciones son:
 - f, --file ARCHIVO Especifica el archivo de composición alternativo a los predeterminados (compose.yml ó docker-compose.yml). Cuando proporciona varios archivos, Compose los combina en una sola configuración. Compose crea la configuración en el orden en que proporciona los archivos.
 - p, --project-name NOMBRE Especifica un nombre de proyecto alternativo predeterminado: el nombre del directorio base
 - project-directory PATH Especifica el directorio de trabajo alternativo
 - profile NOMBRE Especifica un perfil para habilitar
 - e variable=valor Especifica el valor de una variable de entorno

© JMA 2020. All rights reserved

Gestionar composiciones

- Para construir o reconstruir servicios sin crear contenedores:
 - docker compose build
- Para los contenedores de los servicios:
 - docker compose create
- Para crear e iniciar contenedores:
 - docker compose up
- Para detener los contenedores de los servicios:
 - docker compose stop
 - docker compose kill
- Para arrancar los servicios si los contenedores están creados:
 - docker compose start

© JMA 2020. All rights reserved

Gestionar composiciones

- Para parar y arrancar los contenedores de los servicios:
 - docker compose restart
- Para parar los contenedores y borrar los contenedores y redes definidas, con --volumes también borra los volúmenes definidos:
 - docker compose down
- Para obtener la lista de imágenes utilizadas por los contenedores creados:
 - docker compose images
- Para ver la salida de los contenedores de la composición
 - docker compose logs
- Para obtener la lista de proyectos de composición en ejecución
 - docker compose ls

© JMA 2020. All rights reserved

ORQUESTACIÓN

© JMA 2020. All rights reserved

Introducción

- La portabilidad y reproducibilidad de un proceso en contenedores brinda la oportunidad de mover y escalar nuestras aplicaciones en contenedores a través de nubes y centros de datos. Los contenedores garantizan de manera efectiva que esas aplicaciones se ejecuten de la misma manera en cualquier lugar, lo que nos permite aprovechar rápida y fácilmente todos estos entornos.
- Además, a medida que ampliamos nuestras aplicaciones, necesitamos algunas herramientas para ayudar a automatizar el mantenimiento de esas aplicaciones, habilitar el reemplazo automático de contenedores fallidos y administrar la implementación de actualizaciones y reconfiguraciones de esos contenedores durante su ciclo de vida.
- Las herramientas para administrar, escalar y mantener aplicaciones en contenedores se denominan orquestadores, y los ejemplos más comunes de estos son Kubernetes y Docker Swarm. Docker Desktop proporciona las implementaciones del entorno de desarrollo de estos dos orquestadores.

© JMA 2020. All rights reserved



© JMA 2020. All rights reserved

CI INSTALACIÓN (JENKINS Y SONAR)

© JMA 2020. All rights reserved

Contenedores

- Crear versión de Jenkins con Maven, Node y Docker, creando el fichero llamado Dockerfile:

```
FROM jenkins/jenkins:lts
USER root
RUN apt-get update && apt-get install -y lsb-release maven
RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
  https://download.docker.com/linux/debian/gpg
RUN curl -fsSL https://deb.nodesource.com/setup_18.x | bash - && apt-get install -y nodejs
RUN echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
  https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
RUN jenkins-plugin-cli --plugins "blueocean docker-workflow jacoco htmlpublisher sonar job-dsl \
  maven-plugin pipeline-maven"
```

© JMA 2020. All rights reserved

Contenedores

- Generar imagen Docker:
 - docker build -t jenkins-maven-docker .
- Configurar red docker
 - docker network create jenkins
- Crear contenedores Docker
 - docker run -d --name sonarQube --publish 9000:9000 --network jenkins sonarqube:latest
 - docker run -d --name jenkins-docker-in-docker --detach --privileged --network jenkins --network-alias docker --env DOCKER_TLS_CERTDIR=/certs --volume jenkins-docker-certs:/certs/client --volume jenkins-data:/var/jenkins_home --publish 2376:2376 docker:dind --storage-driver overlay2
 - docker run --name jenkins --network jenkins --env DOCKER_HOST=tcp://docker:2376 --env DOCKER_CERT_PATH=/certs/client --env DOCKER_TLS_VERIFY=1 --publish 50080:8080 --publish 50000:50000 --volume \${pwd}/jenkins_home:/home --volume jenkins-data:/var/jenkins_home --volume jenkins-docker-certs:/certs/client:ro --env JAVA_OPTS="--DHUDSON_PLUGINS=git.GitSCM.ALLOW_LOCAL_CHECKOUT=true" jenkins-maven-docker
 - Copiar clave generada para proceder a la instalación (/var/jenkins_home/secrets/initialAdminPassword)
 - Sustituir \${pwd} por %cd% en Windows

© JMA 2020. All rights reserved

MailHog

- MailHog es una herramienta de pruebas de correo electrónico de código abierto dirigida principalmente a los desarrolladores.
- MailHog sirve para emular un servidor de SMTP en local y permite atrapar el correo SMTP saliente, de modo que se puedan ver el contenido de los email sin que estos se envíen realmente.
- Instalación:
 - docker run -d --name mailhog -p 1025:1025 -p 8025:8025 mailhog/mailhog
- Para acceder a la cache del correo saliente:
 - <http://localhost:8025/>

© JMA 2020. All rights reserved

Red e Instalación de Jenkins

- Configurar red docker
 - docker network create jenkins
 - docker network connect jenkins jenkins
 - docker network connect --alias docker jenkins jenkins-docker-in-docker
 - docker network connect jenkins sonarQube
 - docker network connect jenkins mailhog
- En el navegador: <http://localhost:50080/>
 - Desbloquear Jenkins con la clave copiada (docker logs jenkins)
 - Install suggested plugins
 - Create First Admin User
 - Instance Configuration Jenkins URL: <http://localhost:50080>
 - Save and Finish
 - Start using Jenkins

© JMA 2020. All rights reserved

Instalación de Plugins en Jenkins

- Administrar Jenkins > Instalar Plugins
 - JaCoCo
 - Html Publisher
 - SonarQube Scanner
 - Maven Integration
 - Pipeline Maven Integration
 - Docker Pipeline
 - Job DSL
 - Blue Ocean
- Re arrancar al terminar

© JMA 2020. All rights reserved

Configuración de SonarQube

- Entrar SonarQube: <http://localhost:9000>
 - Usuario: admin
 - Contraséña: admin
- Configuración
 - Pasar a castellano: Setting > Marketplace > Plugins: Spanish Pack
 - Los webhooks se utilizan para notificar a servicios externos cuando el análisis de un proyecto ha finalizado. Setting > Webhooks
 - Name: jenkins-webhook
 - URL: http://host.docker.internal:50080/sonarqube-webhook
- Generar un user token en Usuario > My Account > Security
 - Generate Tokens: Jenkins,
 - Type: User Token
 - Generar y copiar

© JMA 2020. All rights reserved

Configuración de Plugins en Jenkins

- Administrar Jenkins > Manage Credentials
 - Click link (global) in System table Global credentials (unrestricted).
 - Click Add credentials
 - Kind: Secret Text
 - Scope: Global
 - Secret: Pegar el token generado en SonarQube
 - Description: Sonar Token
- Administrar Jenkins > Configurar el Sistema
 - En SonarQube servers:
 - Add SonarQube
 - Name: SonarQubeDockerServer (estrictamente el mismo usado en Jenkinsfile)
 - URL del servidor: <http://sonarQube:9000> (el mismo de la instalación docker)
 - Server authentication token: Sonar Token
 - Guardar

© JMA 2020. All rights reserved

Configuración de Plugins en Jenkins

- Administrar Jenkins > Configurar el Sistema
 - En Notificación por correo electrónico
 - Servidor de correo saliente (SMTP): host.docker.internal
 - Puerto de SMTP: 1025
- Administrar Jenkins > Global Tool Configuration
 - Instalaciones de Maven → Añadir Maven
 - Nombre: maven-plugin
 - Instalar automáticamente > Instalar desde Apache
 - Instalaciones de SonarQube Scanner → Añadir SonarQube Scanner
 - Nombre: SonarQube Scanner
 - Instalar automáticamente > Install from Maven Central

© JMA 2020. All rights reserved

Personalizar el contenedor

- Para entrar en modo administrativo
 - docker container exec -u 0 -it jenkins /bin/bash
- Para actualizar la versión de Jenkins
 - mv ./jenkins.war /usr/share/jenkins/
 - chown jenkins:jenkins /usr/share/jenkins/jenkins.war
- Para instalar componentes adicionales
 - apt-get update && apt-get install -y musl-dev
 - ln -s /usr/lib/x86_64-linux-musl/libc.so /lib/libc.musl-x86_64.so.1

© JMA 2020. All rights reserved

<http://www.sonarqube.org/>

SONARQUBE

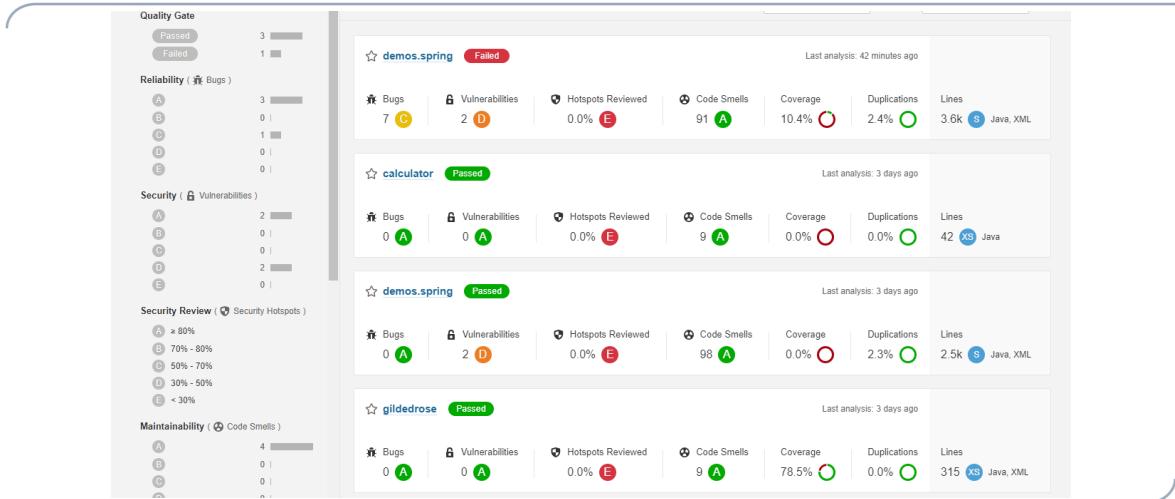
© JMA 2020. All rights reserved

Introducción

- SonarQube (conocido anteriormente como Sonar) es una herramienta de revisión automática de código para detectar errores, vulnerabilidades y malos olores en su código. Puede integrarse con su flujo de trabajo existente para permitir la inspección continua de código en las ramas de su proyecto y las solicitudes de incorporación de cambios.
- SonarQube es una plataforma para la revisión y evaluación del código fuente. Es una herramienta esencial para la fase de pruebas y auditoría de código dentro del ciclo de desarrollo de una aplicación y se considera perfecta para guiar a los equipos de desarrollo durante las revisiones de código.
- Es open source y realiza el análisis estático de código fuente integrando las mejores herramientas de medición de la calidad de código como Checkstyle, PMD o FindBugs, para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.
- Informa sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad ciclomática, posible errores, comentarios y diseño del software.
- Aunque pensado para Java, acepta mas de 20 lenguajes mediante extensiones. Se integra con Maven, Ant y herramientas de integración continua como Atlassian, Jenkins y Hudson.

© JMA 2020. All rights reserved

Sonar



© JMA 2020. All rights reserved

Proceso

1. Los desarrolladores desarrollan y combinan código en un IDE (preferiblemente usando SonarLint para recibir comentarios inmediatos en el editor) y registran su código en su plataforma DevOps.
2. La herramienta de integración continua (CI) de una organización verifica, crea y ejecuta pruebas unitarias, y un escáner SonarQube integrado analiza los resultados.
3. El escáner publica los resultados en el servidor de SonarQube, que proporciona comentarios a los desarrolladores a través de la interfaz de SonarQube, correo electrónico, notificaciones en el IDE (a través de SonarLint) y decoración en las solicitudes de extracción o combinación (cuando se usa Developer Edition y superior).



© JMA 2020. All rights reserved

Beneficios

- Alerta de manera automática a los desarrolladores de los errores de código para corregirlos previamente a la implementación en producción.
- No sólo muestra los errores, también las reglas de codificación, la cobertura de las pruebas, las duplicaciones, la complejidad y la arquitectura, plasmando todos estos datos en paneles de control detallados.
- Ayuda al equipo a mejorar en sus habilidades como programadores al facilitar un seguimiento de los problemas de calidad.
- Permite la creación de paneles y filtros personalizables para centrarse en áreas clave y entregar productos de calidad a tiempo.
- Favorece la productividad al reducir la complejidad del código acortando tiempos y costes adicionales al evitar cambiar el código constantemente.

© JMA 2020. All rights reserved

Herramientas

- [SonarLint](#): es un producto complementario que funciona en su editor y brinda comentarios inmediatos para que pueda detectar y solucionar problemas antes de que lleguen al repositorio.
- [Quality Gate](#): permite saber si su proyecto está listo para la producción.
- [Clean as You Code](#): es un enfoque de la calidad del código que elimina muchos de los desafíos que conllevan los enfoques tradicionales. Como desarrollador, se enfoca en mantener altos estándares y asumir la responsabilidad específicamente en el Nuevo Código en el que está trabajando.
- [Issues](#): SonarQube plantea problemas cada vez que una parte de su código infringe una regla de codificación, ya sea un error que romperá su código (bug), un punto en su código abierto a ataques (vulnerabilidad) o un problema de mantenimiento (código apestoso).
- [Security Hotspots](#): Puntos de acceso de seguridad destaca piezas de código sensibles a la seguridad que deben revisarse. Tras la revisión, descubrirá que no hay ninguna amenaza o que debe aplicar una solución para proteger el código.

© JMA 2020. All rights reserved

Problemas

The screenshot shows a static code analysis interface. On the left, there are filters for Type (Bug: 0, Vulnerability: 0, Code Smell: 8), Severity (Blocker: 0, Critical: 0, Major: 5), Scope, Resolution, Status, Security Category, Creation Date, Language, and Rule. The main area displays issues from pom.xml and src/main/java/com/gildedrose/Item.java.

- pom.xml**
 - Remove this commented out code. Why is this an issue? (3 months ago, L90, Code Smell, Major, Open, Not assigned, 5min effort, Comment, unused)
 - This block of commented-out lines of code should be removed. Why is this an issue? (3 months ago, L118, Code Smell, Major, Open, Not assigned, 5min effort, Comment, unused)
 - This block of commented-out lines of code should be removed. Why is this an issue? (3 months ago, L130, Code Smell, Major, Open, Not assigned, 5min effort, Comment, unused)
- src/main/java/com/gildedrose/Item.java**
 - Make name a static final constant or non-public and provide accessors if needed. Why is this an issue? (3 months ago, L4, Code Smell, Minor, Open, Not assigned, 10min effort, Comment, cwe)
 - Make sellIn a static final constant or non-public and provide accessors if needed. Why is this an issue? (3 months ago, L6, Code Smell, Minor, Open, Not assigned, 10min effort, Comment, cwe)
 - Make quality a static final constant or non-public and provide accessors if needed. Why is this an issue? (3 months ago, L8, Code Smell, Minor, Open, Not assigned, 10min effort, Comment, cwe)

© JMA 2020. All rights reserved

Puntos de acceso de seguridad

The screenshot shows a security review interface. At the top, it says "Filters Assigned to me All", "Status To review", "Overall code", and "Security Hotspots Reviewed 0.0%". It also shows a "Review priority: LOW".

1 Security Hotspots to review

Insecure Configuration (1)
Make sure this debug feature is deactivated before delivering the code in production.
Delivering code in production with debug features activated is security-sensitive. java:S4507
Add Comment Open in IDE Get Permalink

Category: Insecure Configuration
Review priority: LOW
Assignee: Not assigned

Status: To review
This Security Hotspot needs to be reviewed to assess whether the code poses a risk
Change status

src/main/java/com/gildedrose/Item.java

```
11     this.name = name;
12     this.sellIn = sellIn;
13     try {
14         setQuality(quality);
15     } catch (Exception e) {
16         e.printStackTrace();
17     }
18
19
20     public String getName() {
21         return name;
22     }
```

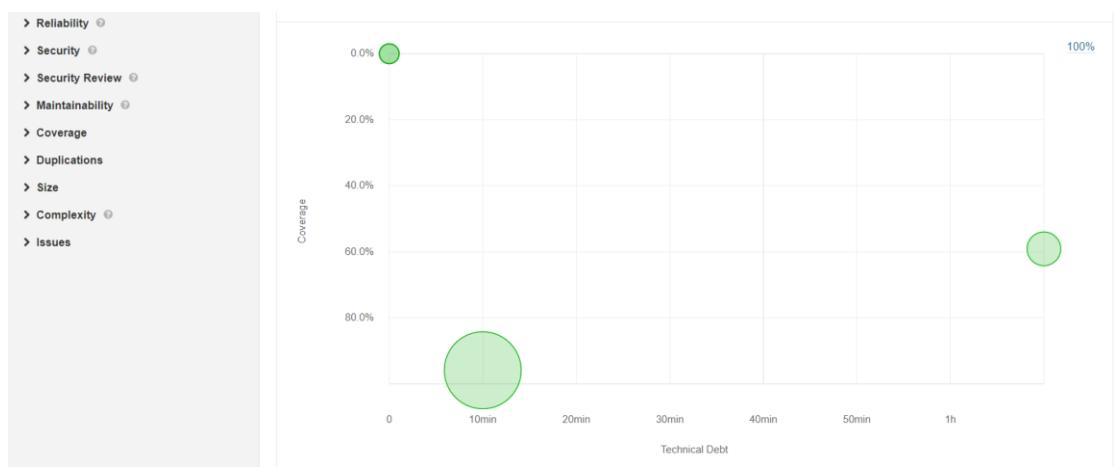
© JMA 2020. All rights reserved

Métricas principales

- **Complejidad:** Refleja la Complejidad Ciclomática calculada en base al número de caminos a través del código normalmente observado a nivel de métodos o funciones individuales.
- **Duplicados:** Nos indica el número de bloques de líneas duplicados. Ayuda a evitar resultados distintos en operaciones iguales.
- **Evidencias:** Son los fragmentos nuevos de código de un proyecto que detecta que incumplen con alguna de las reglas establecidas.
- **Mantenibilidad:** Se refiere al recuento total de problemas de Code Smell.
- **Umbrales de calidad:** Define los requisitos del proyecto antes de ser lanzado a producción, como, por ejemplo, que no deben haber evidencias bloqueantes o la cobertura de código sobre el código nuevo debe ser mayor que el 80%.
- **Tamaño:** Permiten hacerse una idea del volumen del proyecto en términos generales.
- **Pruebas:** Son una forma de comprobar el correcto funcionamiento de una unidad de código y de su integración.

© JMA 2020. All rights reserved

Métricas



© JMA 2020. All rights reserved

Análisis

- SonarQube puede analizar mas de 20 lenguajes diferentes según la edición. El resultado de este análisis serán métricas y problemas de calidad (casos en los que se rompieron las reglas de codificación). Sin embargo, lo que se analiza variará según el lenguaje:
 - En todos los lenguajes, los datos de "culpa" se importarán automáticamente de los proveedores de SCM admitidos. Git y SVN son compatibles automáticamente.
 - En todos los lenguajes se realiza un análisis estático del código fuente (archivos Java, programas COBOL, etc.)
 - Se puede realizar un análisis estático del código compilado para ciertos lenguajes (archivos .class en Java, archivos .dll en C#, etc.)
- Durante el análisis, se solicitan datos del servidor, se analizan los archivos proporcionados para el análisis enfrentándolos a las reglas y los datos resultantes se envían de vuelta al servidor al final en forma de informe, que luego se analiza de forma asíncrona en el lado del servidor.

© JMA 2020. All rights reserved

Reglas

- Una regla es un estándar o práctica de codificación que debe seguirse. El incumplimiento de las reglas de codificación conduce a errores, vulnerabilidades, puntos críticos de seguridad y código apestoso. Las reglas pueden comprobar la calidad de los archivos de código o las pruebas unitarias.
- Cada lenguaje de programación dispone de sus propias reglas, siendo diferentes en cantidad y tipo.
- Las reglas se clasifican en:
 - Bug: un punto de fallo real o potencial en su software
 - Code Smell: un problema relacionado con la mantenibilidad en el código.
 - Vulnerability: un punto débil que puede convertirse en un agujero de seguridad que puede usarse para atacar su software.
 - Security Hotspot: un problema que es un agujero de seguridad.
- Para Bugs y Code Smells and Bugs, no se esperan falsos positivos. Al menos este es el objetivo para que los desarrolladores no tengan que preguntarse si se requiere una solución. Para las vulnerabilidades, el objetivo es que más del 80 % de los problemas sean verdaderos positivos. Las reglas de Security Hotspot llaman la atención sobre el código que es sensible a la seguridad. Se espera que más del 80% de los problemas se resuelvan rápidamente como "Revisados" después de la revisión por parte de un desarrollador.

© JMA 2020. All rights reserved

Control de calidad

- Los perfiles de calidad (Quality Profile) son un componente central de SonarQube donde se definen conjuntos de reglas que, cuando se violan, generan problemas en la base de código (ejemplo: los métodos no deben tener una complejidad cognitiva superior a 15). Cada lenguaje individual tiene su propio perfil de calidad predeterminado y los proyectos que no están asignados explícitamente a perfiles de calidad específicos se analizan utilizando los perfiles de calidad predeterminados.
- Un umbral de calidad (Quality Gate) es un conjunto de condiciones booleanas basadas en métricas. Ayudan a saber inmediatamente si los proyectos están listos para la producción. Idealmente, todos los proyectos utilizarán la misma barrera de calidad. El estado de Quality Gate de cada proyecto se muestra de forma destacada en la página de inicio.

© JMA 2020. All rights reserved

Scanner

- Una vez que se haya instalado la plataforma SonarQube, estará listo para instalar un escáner y comenzar a crear proyectos. Para ello, se debe instalar y configurar el escáner más adecuado para el proyecto:
 - Gradle - SonarScanner para Gradle
 - Ant - SonarScanner para Ant
 - Maven: use el SonarScanner para Maven
 - .NET - SonarScanner para .NET
 - Jenkins - SonarScanner para Jenkins
 - Azure DevOps - Extensión de SonarQube para Azure DevOps
 - Cualquier otra cosa (CLI) - SonarScanner

© JMA 2020. All rights reserved

SonarScanner para Maven

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.10.0</version>
    </plugin>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.9.1.2184</version>
    </plugin>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.8.7</version>
    </plugin>
  </plugins>
</build>
```

© JMA 2020. All rights reserved

SonarScanner para Maven

```
<profiles>
  <profile>
    <id>coverage</id><activation><activeByDefault>true</activeByDefault></activation>
    <build>
      <plugins>
        <plugin>
          <groupId>org.jacoco</groupId>
          <artifactId>jacoco-maven-plugin</artifactId>
          <executions>
            <execution>
              <id>prepare-agent</id><goals><goal>prepare-agent</goal></goals>
            </execution>
            <execution>
              <id>report</id><goals><goal>report</goal></goals>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

© JMA 2020. All rights reserved

Analizando

- Analizar un proyecto Maven consiste en ejecutar un objetivo Maven: sonar:sonar desde el directorio que contiene el proyecto principal pom.xml.
- Se debe pasar un token de autenticación usando la propiedad sonar.login en la línea de comando.
 - mvn sonar:sonar -Dsonar.projectKey=Microservicios -Dsonar.host.url=http://localhost:9000 -Dsonar.login=1eeaaf436541...
- Para obtener información de cobertura, deberá generar el informe de cobertura antes del análisis.
 - mvn clean verify sonar:sonar

© JMA 2020. All rights reserved

JavaScript

- El SonarScanner es el escáner que se debe usar cuando no hay un escáner específico para su sistema de compilación. Para descargar:
 - <https://docs.sonarqube.org/8.9/analysis/scan/sonarscanner/>
- Crear un archivo de configuración en el directorio raíz del proyecto llamado sonar-project.properties
 - sonar.projectKey=<projectKey>
 - sonar.projectName=< projectName >
 - sonar.projectVersion=1.0
 - sonar.language=js
 - sonar.sources=src
 - sonar.sourceEncoding=UTF-8
 - #sonar.host.url=http://localhost:9000
- Para ejecutar el scanner:
 - sonar-scanner.bat -D"sonar.projectKey=Web4Testing" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.login=9ea...51a3a"

© JMA 2020. All rights reserved

JENKINS

© JMA 2020. All rights reserved

Introducción

- Jenkins es un servidor open source para la integración continua. Es una herramienta que se utiliza para compilar y probar proyectos de software de forma continua, lo que facilita a los desarrolladores integrar cambios en un proyecto y entregar nuevas versiones a los usuarios. Escrito en Java, es multiplataforma y accesible mediante interfaz web. Es el software más utilizado en la actualidad para este propósito.
- Con Jenkins, las organizaciones aceleran el proceso de desarrollo y entrega de software a través de la automatización. Mediante sus centenares de plugins, se puede implementar en diferentes etapas del ciclo de vida del desarrollo, como la compilación, la documentación, el testeo o el despliegue.
- La primera versión de Jenkins surgió en 2011, pero su desarrollo se inició en 2004 como parte del proyecto Hudson. Kohsuke Kawaguchi, un desarrollador de Java que trabajaba en Sun Microsystems, creó un servidor de automatización para facilitar las tareas de compilación y de realización de pruebas.
- En 2010, surgieron discrepancias relativas a la gestión del proyecto entre la comunidad y Oracle y, finalmente, se decidió el cambio de denominación a "Jenkins". Desde entonces los dos proyectos continuaron desarrollándose independientemente. Hasta que finalmente Jenkins se impuso al ser utilizado en muchos más proyectos y contar con más contribuyentes.

© JMA 2020. All rights reserved

Por qué usarlo

- Antes de disponer de herramientas como Jenkins para poder aplicar integración continua nos encontrábamos con un escenario en el que:
 - Todo el código fuente era desarrollado y luego testeado, con lo que los despliegues y las pruebas eran muy poco habituales y localizar y corregir errores era muy laborioso. El tiempo de entrega del software se prolongaba.
 - Los desarrolladores tenían que esperar al desarrollo de todo el código para poner a prueba sus mejoras.
 - El proceso de desarrollo y testeo eran manuales, por lo que era más probable que se produjeran fallos.
- Sin embargo, con Jenkins (y la integración continua que facilita) la situación es bien distinta:
 - Cada commit es desarrollado y verificado. Con lo que en lugar de comprobar todo el código, los desarrolladores sólo necesitan centrarse en un commit concreto para corregir bugs.
 - Los desarrolladores conocen los resultados de las pruebas de sus cambios durante la ejecución.
 - Jenkins automatiza las pruebas y el despliegue, lo que ahorra mucho tiempo y evita errores.
 - El ciclo de desarrollo es más rápido. Se entregan más funcionalidades y más frecuentemente a los usuarios, con lo que los beneficios son mayores.

© JMA 2020. All rights reserved

Qué se puede hacer

- Con Jenkins podemos automatizar multitud de tareas que nos ayudarán a reducir el time to market de nuestros productos digitales o de nuevas versiones de ellos. Concretamente, con esta herramienta podemos:
 - Automatizar la compilación y testeo de software.
 - Notificar a los equipos correspondientes la detección de errores.
 - Desplegar los cambios en el código que hayan sido validados.
 - Hacer un seguimiento de la calidad del código y de la cobertura de las pruebas.
 - Generar la documentación de un proyecto.
 - Podemos ampliar las funcionalidades de Jenkins a través de múltiples plugins creados por la comunidad, diseñados para ayudarnos en centenares de tareas, a lo largo de las diferentes etapas del proceso de desarrollo.

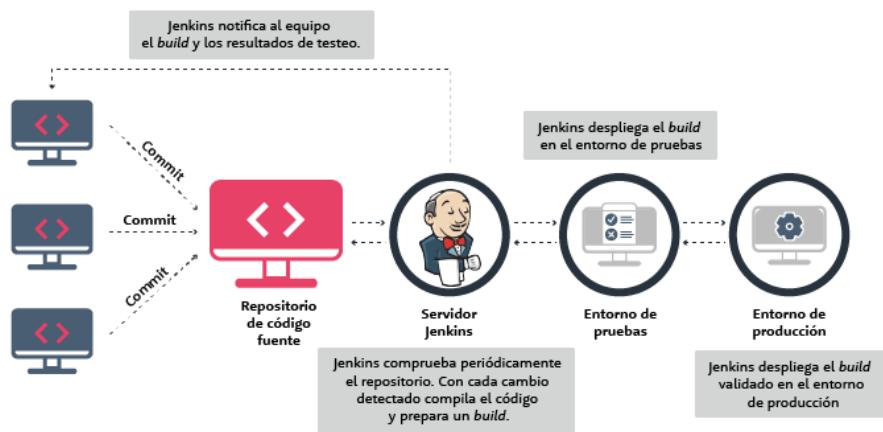
© JMA 2020. All rights reserved

Cómo funciona

- Para entender cómo funciona Jenkins vamos a ver un ejemplo de cómo sería el flujo de integración continua utilizando esta herramienta:
 1. Un desarrollador hace un commit de código en el repositorio del código fuente.
 2. El servidor de Jenkins hace comprobaciones periódicas para detectar cambios en el repositorio.
 3. Poco después del commit, Jenkins detecta los cambios que se han producido en el código fuente. Compila el código y prepara un build. Si el build falla, envía una notificación al equipo en cuestión. Si resulta exitoso, lo despliega en el servidor de testeо.
 4. Después de la prueba, Jenkins genera un feedback y notifica al equipo el build y los resultados del testeо.
 5. Jenkins continúa revisando el repositorio frecuentemente y todo el proceso se repite.

© JMA 2020. All rights reserved

Cómo funciona



© JMA 2020. All rights reserved

Pros y Contras

- Ventajas
 - Es sencilla de instalar.
 - Es una herramienta opensource respaldada por una gran comunidad.
 - Es gratuita.
 - Es muy versátil, gracias a sus centenares de plugins.
 - Está desarrollada en Java, por lo que funciona en las principales plataformas.
- Desventajas
 - Su interfaz de usuario es anticuada y poco intuitiva, aunque puede mejorarse con plugins como Blue Ocean.
 - Sus pipelines son complejas y pueden requerir mucho tiempo de dedicación a las mismas.
 - Algunos de sus plugins están desfasados.
 - Necesita de un servidor de alojamiento, que puede conllevar configuraciones tediosas y requerir ciertos conocimientos técnicos.
 - Necesita ampliar su documentación en algunas áreas.

© JMA 2020. All rights reserved

Proyectos

- Un proyecto de construcción de Jenkins contiene la configuración para automatizar una tarea o paso específico en el proceso de creación de aplicaciones. Estas tareas incluyen recopilar dependencias, compilar, archivar o transformar código y probar e implementar código en diferentes entornos.
- Jenkins admite varios tipos de trabajos de compilación
 - Proyecto de estilo libre (freestyle): Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.
 - Proyecto Maven: Ejecuta un proyecto maven. Jenkins es capaz de aprovechar la configuración presente en los ficheros POM, reduciendo drásticamente la configuración.
 - Pipeline: Gestiona actividades de larga duración que pueden abarcar varios agentes de construcción. Apropiado para construir pipelines (conocidas anteriormente como workflows) y/o para la organización de actividades complejas que no se pueden articular fácilmente con tareas de tipo freestyle.
 - Proyecto multi-configuration: Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.
 - Carpeta: Crea un contenedor que almacena elementos anidados en él..
 - Multibranch Pipeline: Crea un conjunto de proyectos Pipeline según las ramas detectadas en un repositorio de SCM.

© JMA 2020. All rights reserved

Freestyle Project

- Jenkins Freestyle Project es un trabajo de construcción, secuencia de comandos o canalización repetible que contiene pasos y acciones posteriores a la construcción. Es un trabajo o tarea mejorado que puede abarcar múltiples operaciones. Permite configurar activadores de compilación y ofrece seguridad basada en proyectos para un proyecto de Jenkins. También ofrece complementos para ayudarlo a construir pasos y acciones posteriores a la construcción.
- Aunque los trabajos de estilo libre son muy flexibles, admiten un número limitado de acciones generales de compilación y posteriores a la compilación. Cualquier acción especializada o no típica que un usuario quiera agregar a un proyecto de estilo libre requiere complementos adicionales.
- Los elementos de este trabajo son:
 - SCM opcional, como CVS o Subversion donde reside el código fuente.
 - Disparadores opcionales para controlar cuándo Jenkins realizará compilaciones.
 - Algun tipo de script de construcción que realiza la construcción (ant, maven, script de shell, archivo por lotes, etc.) donde ocurre en trabajo real.
 - Pasos opcionales para recopilar información de la construcción, como archivar los artefactos y/o registrar javadoc y resultados de pruebas.
 - Pasos opcionales para notificar a otras personas/sistemas con el resultado de la compilación, como enviar correos electrónicos, mensajes instantáneos, actualizar el rastreador de problemas, etc.

© JMA 2020. All rights reserved

Configurar un trabajo

- Desechar ejecuciones antiguas determina cuándo se deben descartar los registros de compilación para este proyecto. Los registros de compilación incluyen la salida de la consola, los artefactos archivados y cualquier otro metadato relacionado con una compilación en particular. Mantener menos compilaciones significa que se usará menos espacio en disco en el directorio raíz del registro.
- Los parámetros permiten solicitar a los usuarios una o más entradas que se pasarán a una compilación. Cada parámetro tiene un Nombre y algún tipo de Valor, dependiendo del tipo de parámetro. Estos pares de nombre y valor se exportarán como variables de entorno cuando comience la compilación, lo que permitirá que las partes posteriores de la configuración de la compilación (como los pasos de la compilación) accedan a esos valores, usando la sintaxis \${PARAMETER_NAME}.
- El plugins de git proporciona operaciones fundamentales de git para los proyectos de Jenkins. Puede sondear, buscar, pagar y fusionar contenidos de repositorios de git.
- Los disparadores de ejecuciones automatizan la ejecución del proyecto: Lanzar ejecuciones remotas (ejem: desde 'scripts'), Construir tras otros proyectos, Ejecutar periódicamente, GitHub hook trigger for GITScm polling, Consultar repositorio (SCM)

© JMA 2020. All rights reserved

Configurar un trabajo

- Desechar ejecuciones antiguas determina cuándo se deben descartar los registros de compilación para este proyecto. Los registros de compilación incluyen la salida de la consola, los artefactos archivados y cualquier otro metadato relacionado con una compilación en particular. Mantener menos compilaciones significa que se usará menos espacio en disco en el directorio raíz del registro.
- Los parámetros permiten solicitar a los usuarios una o más entradas que se pasarán a una compilación. Cada parámetro tiene un Nombre y algún tipo de Valor, dependiendo del tipo de parámetro. Estos pares de nombre y valor se exportarán como variables de entorno cuando comience la compilación, lo que permitirá que las partes posteriores de la configuración de la compilación (como los pasos de la compilación) accedan a esos valores, usando la sintaxis \${PARAMETER_NAME}.
- El plugins de git proporciona operaciones fundamentales de git para los proyectos de Jenkins. Puede sondear, buscar, pagar y fusionar contenidos de repositorios de git.
- Los disparadores de ejecuciones automatizan la ejecución del proyecto: Lanzar ejecuciones remotas (ejem: desde 'scripts'), Construir tras otros proyectos, Ejecutar periódicamente, GitHub hook trigger for GITScm polling, Consultar repositorio (SCM)

© JMA 2020. All rights reserved

Configurar un trabajo

- En la ejecución se indica el paso o conjunto de pasos a dar por Jenkins para realizar la construcción de la aplicación. Los pasos pueden ser: ejecutar en línea de comandos, ejecutar Ant o Gradle, ejecutar tareas 'maven' de nivel superior, procesar trabajos DSLs, ... Los pasos disponibles dependen de los plugins instalados.
- Se pueden establecer acciones para ejecutar después de la ejecución de los pasos: Notificación por correo, Editable Email Notification, Ejecutar otros proyectos, Guardar los archivos generados, Agregar los resultados de los tests de los proyectos padre, Almacenar firma de ficheros para poder hacer seguimiento, Publicar Javadoc, Publicar los resultados de tests JUnit, Publish HTML reports, Git Publisher, Set GitHub commit status (universal), Delete workspace when build is done.

© JMA 2020. All rights reserved

Notificación por correo

- Una de las acciones para ejecutar después más típica es enviar un correo electrónico para cada compilación inestable. De esta forma se notifican rápidamente a los desarrolladores de los problemas ocurridos.
- Si está configurada la acción de Notificación por correo en el trabajo, Jenkins enviará un correo electrónico a los destinatarios especificados cuando ocurra un determinado evento importante:
 - Cada compilación fallida desencadena un nuevo correo electrónico.
 - Una compilación exitosa después de una compilación fallida (o inestable) activa un nuevo correo electrónico, lo que indica que la crisis ha terminado.
 - Una compilación inestable después de una compilación exitosa desencadena un nuevo correo electrónico, lo que indica que hay una regresión.
 - A menos que se configure, cada compilación inestable desencadena un nuevo correo electrónico, lo que indica que la regresión todavía está allí.
- La Notificación por correo requiere tener instalado el plugin Mailer y configurarlo en Manage Jenkins > Configure System > E-mail Notification.

© JMA 2020. All rights reserved

Jenkins DSL

- El plugin DSL permite crear varios jobs de forma automática a partir de un job principal llamado seed (semilla o plantilla). De esta forma evitamos tener que volver a crear los mismos jobs para cada nuevo proyecto que precise de tareas similares, pudiendo montar todas las tareas para cada proyecto simplemente ejecutando la tarea semilla. Si modificamos la tarea semilla y volvemos a ejecutarla Jenkins, actualizará los jobs ya creados.
- La tarea semilla es un proyecto de estilo libre que consta básicamente un paso Process Job DSLs de ejecución con un script de Groovy donde se definen la configuración de las tareas a crear de forma automática: mediante instrucciones se configuran las opciones del proyecto que anteriormente se configuraban manualmente a través del interfaz gráfico.
- Es necesario habilitar los scripts cada vez que se modifican:
 - Administrar Jenkins > In-process Script Approval

© JMA 2020. All rights reserved

Jenkins DSL

```
job('generado-por-dsl') {
    description('La tarea se ha generado desde otro job')
    parameters {
        booleanParam('FLAG', true)
        choiceParam('OPTION', ['option 1 (default)', 'option 2', 'option 3'])
    }
    scm {
        github('jenkins-docs/simple-java-maven-app', 'master')
    }
    steps {
        shell("echo 'Empiezo el proceso'")
        maven('clean verify')
    }
    publishers {
        mailer('me@example.com', true, true)
    }
}
```

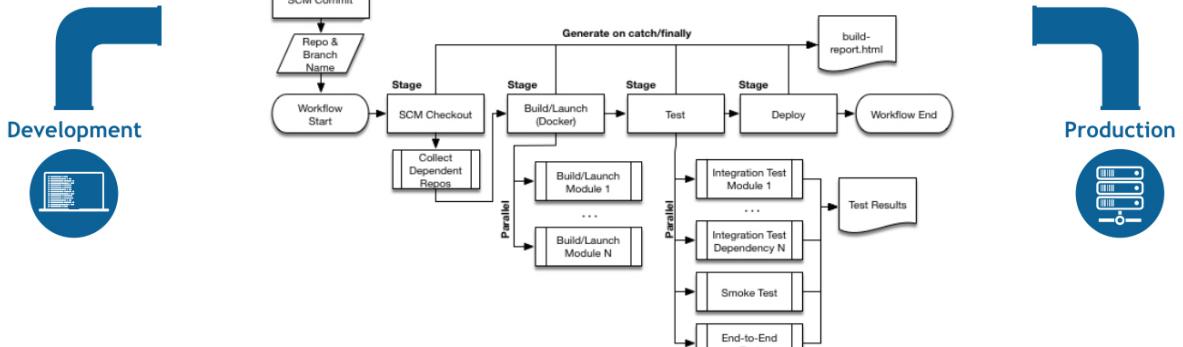
© JMA 2020. All rights reserved

Pipeline

- Un pipeline es una forma de trabajar en el mundo DevOps, bajo la Integración Continua, que nos permite definir el ciclo de vida completo de un desarrollo de software.
- Un pipeline consiste en un flujo comprendido en varias fases que van en forma secuencial, siendo la entrada de cada una la salida de la anterior. Es un conjunto de instrucciones del proceso que sigue una aplicación desde el repositorio de control de versiones hasta que llega a los usuarios.
- El pipeline estaría formado por un conjunto de procesos o herramientas automatizadas que permiten que tanto los desarrolladores como otros roles, trabajen de forma coherente para crear e implementar código en un entorno de producción.
- Cada cambio en el software, lleva a un complejo proceso hasta que es desplegado. Este proceso incluye desde construir software de forma fiable y repetible (conocido como “build”), hasta realizar todos los pasos de testing y los despliegues necesarios.
- Un pipeline Jenkins es un conjunto de plugins que soporta la implementación e integración de pipelines (canalizaciones) de despliegue continuo en Jenkins. Jenkins provee un gran conjunto de herramientas para dar forma con código a un flujo de entrega de la aplicación.

© JMA 2020. All rights reserved

Pipeline



© JMA 2020. All rights reserved

Definición

- La definición de un pipeline Jenkins se escribe en un fichero de texto (llamado Jenkinsfile) que se puede subir al repositorio junto con el resto del proyecto de software.
- Ésta es la base del “Pipeline como código”: tratar la canalización de despliegue continuo como parte de la aplicación para que sea versionado y revisado como cualquier otra parte del código.
- La creación de un Jenkinsfile y su subida al repositorio de control de versiones, otorga una serie de inmediatos beneficios:
 - Crear automáticamente un pipeline de todo el proceso de construcción para todas las ramas y todos los pull request.
 - Revisión del código en el ciclo del pipeline.
 - Auditar todos los pasos a seguir en el pipeline
 - Una sencilla y fiable fuente única, que puede ser vista y editada por varios miembros del proyecto.

© JMA 2020. All rights reserved

Sintaxis de canalización

- Para definir una canalización se puede utilizar la sintaxis de canalización declarativa y la sintaxis de canalización con secuencias de comandos.
- En la sintaxis de canalización declarativa, el bloque pipeline define todo el trabajo realizado a lo largo de todo el Pipeline.

```
pipeline {  
    // ...  
}
```

- En la sintaxis de canalización con secuencias de comandos, uno o más bloques node hacen el trabajo principal en todo el Pipeline. Aunque no es un requisito obligatorio de la sintaxis con secuencias de comandos, confinar el trabajo del Pipeline dentro de un bloque node hace dos cosas:

- Programa los pasos contenidos dentro del bloque para que se ejecuten agregando un elemento a la cola de Jenkins. Tan pronto como un ejecutor esté libre en un nodo, los pasos se ejecutarán.
- Crea un espacio de trabajo (un directorio específico para ese canal en particular) donde se puede trabajar en los archivos extraídos del control de código fuente.

```
node {  
    // ...  
}
```

© JMA 2020. All rights reserved

Sintaxis Básica Declarativa

<https://www.jenkins.io/doc/book/pipeline/syntax/>

- Pipeline {} Identificamos dónde empieza y termina el pipeline así como los pasos que tiene
- Agent. Especificamos cuando se ejecuta el pipeline. Uno de los comandos más utilizados es any, para ejecutar el pipeline siempre y cuando haya un ejecutor libre en Jenkins.
- Node (nodo): Máquina que es parte del entorno de Jenkins y es capaz de ejecutar un Pipeline Jenkins. Los nodos son agrupaciones de tareas o steps que comparten un workspace.
- Stages (etapas). Bloque donde se definen una serie de etapas a realizar dentro del pipeline.
- Stage. Son las etapas lógicas en las que se dividen los flujos de trabajo de Jenkins. Bloque que define una serie de tareas realizadas dentro del pipeline, por ejemplo: build, test, deploy, etc. Podemos utilizar varios plugins en Jenkins para visualizar el estado o el progreso de estos estados.
- Steps (pasos). Son todos los pasos a realizar dentro de un stage. Podemos definir uno o varios pasos.
- Step. Son los pasos lógicos en las que se dividen los flujos de trabajo de Jenkins. Es una tarea simple dentro del pipeline. Fundamentalmente es un paso donde se le dice a Jenkins qué hacer en un momento específico o paso del proceso. Por ejemplo, para ejecutar un comando en shell podemos tener un paso en el que tengamos la línea sh ls para mostrar el listado de ficheros de una carpeta.

© JMA 2020. All rights reserved

Jenkinsfile

```
pipeline {  
    agent any  
    triggers { // Sondear repositorio a intervalos regulares  
        pollSCM('* * * * *')  
    }  
    stages {  
        stage("Compile") {  
            steps {  
                sh "mvn compile"  
            }  
        }  
        stage("Unit test") {  
            steps {  
                sh "mvn test"  
            }  
            post {  
                always {  
                    junit 'target/surefire-reports/*.xml'  
                }  
            }  
        }  
        stage("SonarQube Analysis") {  
            steps {  
                withSonarQubeEnv('SonarQubeDockerServer') {  
                    sh 'mvn clean verify sonar:sonar'  
                }  
                timeout(2) { // time: 2 unit: 'MINUTES'  
                    // In case of SonarQube failure or direct timeout exceed, stop Pipeline  
                    waitForQualityGate abortPipeline: waitForQualityGate().status != 'OK'  
                }  
            }  
        }  
        stage("Build") {  
            steps {  
                sh "mvn package -DskipTests"  
            }  
        }  
        stage("Deploy") {  
            steps {  
                sh "mvn install -DskipTests"  
            }  
        }  
    }  
}
```

© JMA 2020. All rights reserved

Definición de entornos de ejecución

- La arquitectura de Jenkins está diseñada para entornos de construcción distribuidos. Nos permite usar diferentes entornos para cada proyecto de compilación, equilibrando la carga de trabajo entre múltiples agentes que ejecutan trabajos en paralelo.
- El controlador de Jenkins es el nodo original de la instalación de Jenkins. El controlador de Jenkins administra los agentes de Jenkins y organiza su trabajo, incluida la programación de trabajos en agentes y la supervisión de agentes. Los agentes se pueden conectar al controlador Jenkins mediante equipos locales o en la nube.
- Hay varias formas de definir agentes para usar en Pipeline, como máquinas físicas, máquinas virtuales, clústeres de Kubernetes y con imágenes de Docker.
- Los agentes requieren una instalación de Java y una conexión de red al controlador Jenkins.

© JMA 2020. All rights reserved

Definición de entornos de ejecución

- Al ejecutar una canalización:
 - Jenkins pone en cola todos los pasos contenidos en el bloque para que los ejecute. Tan pronto como haya un ejecutor disponible, los pasos comenzarán a ejecutarse.
 - Se asigna un espacio de trabajo que contendrá archivos extraídos del control de versiones, así como cualquier archivo de trabajo adicional para Pipeline.
- La directiva agent, obligatoria para todas las canalizaciones, le dice a Jenkins dónde y cómo ejecutar todo el Pipeline o un subconjunto del mismo (una etapa específica).

```
pipeline {  
    agent any
```

© JMA 2020. All rights reserved

Definición de entornos de ejecución

- La directiva agent permite ejecutar el Pipeline o una etapa:
 - none: Cuando se aplica en el bloque de nivel superior del pipeline, no se asignará ningún agente global para toda la ejecución de Pipeline y cada sección stage deberá contener su propia sección agent. Por ejemplo: agent none
 - any: en cualquier agente disponible. Por ejemplo: agent any
 - label: en un agente definido en el entorno de Jenkins con la etiqueta proporcionada. Se pueden utilizar condiciones de etiqueta. Por ejemplo: agent { label 'my-defined-label' }
 - node: se comporta igual que agent { label 'labelName' }, pero permite opciones adicionales: agent { node { label 'labelName' } }
 - docker: con el contenedor dado que se aprovisionará dinámicamente en un nodo preconfigurado para aceptar Pipelines basados en Docker: agent { docker 'maven:3.8.1-adoptopenjdk-11' }
 - dockerfile: con un contenedor creado a partir de un Dockerfile contenido en el repositorio de origen: agent { dockerfile true }.
 - kubernetes: dentro de un pod implementado en un clúster de Kubernetes.

© JMA 2020. All rights reserved

Disparadores

- La directiva triggers, opcional pero única dentro del bloque pipeline, permite definir localmente las formas automatizadas en las que se debe volver a activar Pipeline. Si están disponibles integraciones basada en webhooks (como GitHub o BitBucket) serán innecesarios. Los activadores actualmente disponibles son:
 - cron: Acepta una cadena de estilo cron para definir un intervalo regular en el que se debe volver a activar:
triggers { cron('H */4 * * 1-5') }
 - pollSCM: Acepta una cadena de estilo cron para definir un intervalo regular en el que Jenkins debe verificar si hay nuevos cambios en el control se versione, en cuyo caso la canalización se volverá a activar:
triggers { pollSCM('H */4 * * 1-5') }
 - upstream: Acepta una lista de trabajos separados por comas y un umbral. Cuando cualquier trabajo en la lista finaliza con el umbral mínimo, la canalización se volverá a activar:
triggers { upstream(upstreamProjects: 'job1,job2', threshold: hudson.model.Result.SUCCESS) }

© JMA 2020. All rights reserved

Variables de entorno

- Las variables de entorno se pueden configurar globalmente o por etapa (solo se aplicarán a la etapa en la que están definidas). Este enfoque para definir variables desde dentro Jenkinsfile puede ser muy útil para compartir valores e instruir scripts, como un Makefile, para configurar la compilación o las pruebas de manera diferente para ejecutarlas dentro de Jenkins.
- La configuración de una variable de entorno dentro de una canalización depende de si se utiliza una canalización declarativa (directiva environment) o con secuencias de comandos (paso withEnv).

```
pipeline {
    agent any
    environment {
        DOCKERHUB_CREDENTIALS = credentials('DockerHub')
    }
    stages {
        stage("SonarQube Analysis") {
            environment {
                scannerHome = tool 'SonarQube Scanner'
            }
            steps {

```

© JMA 2020. All rights reserved

Variables de entorno

- Por convención, los nombres de las variables de entorno suelen especificarse en mayúsculas, con palabras individuales separadas por guiones bajos.
- Las variables de entorno se pueden configurar en tiempo de ejecución y se pueden usar con scripts de shell (sh), por lotes de Windows (bat) y de PowerShell (powershell). Cada script puede devolver el estado (returnStatus) o la salida estándar (returnStdout).

```
environment {
    CC = """${sh(
        returnStdout: true, script: 'echo "clang"'
    )}"""
    EXIT_STATUS = """${sh(
        returnStatus: true, script: 'exit 1'
    )}"""
}
```

- Jenkins Pipeline admite declarar una cadena con comillas simples o comillas dobles, pero la interpolación de cadenas con la notación \${exp} solo está disponible con comillas dobles:
sh("curl https://example.com/doc/\${EXIT_STATUS}")
sh 'echo \$DOCKERHUB_CREDENTIALS_PSW' // sin interpolación

© JMA 2020. All rights reserved

Variables de entorno

- Jenkins Pipeline expone las variables de entorno a través de la variable global env, disponible desde cualquier lugar dentro de un archivo Jenkinsfile.
- Los valores más comúnmente usados son: JOB_NAME, BUILD_ID, BUILD_NUMBER, BUILD_TAG, BUILD_URL, EXECUTOR_NUMBER, JAVA_HOME, JENKINS_URL, NODE_NAME, WORKSPACE.
echo "Running \${env.BUILD_ID} on \${env.JENKINS_URL}"
- La variable global currentBuild se puede usar para hacer referencia a la compilación que se está ejecutando actualmente: id, number, displayName, projectName, description, result (SUCCESS, UNSTABLE, FAILURE, ...), ...
mail to: 'team@example.com',
 subject: "Status of pipeline: \${currentBuild.fullDisplayName}",
 body: "\${env.BUILD_URL} has result \${currentBuild.currentResult}"
- La lista completa está disponible en <http://localhost:50080/pipeline-syntax/globals>.
- Los complementos de Jenkins también pueden suministrar y establecer variables de entorno, los valores disponibles son específicos de cada complemento .

© JMA 2020. All rights reserved

Credenciales

- Otro uso común para las variables de entorno es establecer o anular credenciales "ficticias" en scripts de compilación o prueba. Debido a que es una mala práctica colocar las credenciales directamente en un Jenkinsfile, Jenkins Pipeline permite a los usuarios acceder de forma rápida y segura a las credenciales predefinidas en el Jenkinsfile sin necesidad de conocer sus valores.
- La sintaxis declarativa de Pipeline, dentro de environment, tiene el método de ayuda credentials() que admite credenciales de texto secreto, usuario con contraseña y archivos secretos.

```
environment {  
    DOCKERHUB_CREDENTIALS = credentials('DockerHub')  
}
```
- Si es una credencial usuario con contraseña, establece las siguientes tres variables de entorno:
 - DOCKERHUB_CREDENTIALS: contiene un nombre de usuario y una contraseña separados por dos puntos en el formato username:password.
 - DOCKERHUB_CREDENTIALS_USR: una variable adicional que contiene solo el nombre de usuario.
 - DOCKERHUB_CREDENTIALS_PSW: una variable adicional que contiene solo la contraseña.
- Para mantener la seguridad y el anonimato de estas credenciales, si el trabajo muestra el valor de estas variables de credenciales desde dentro del Pipeline, Jenkins solo devuelve el valor "*****" para reducir el riesgo de que se divulgue información secreta.

© JMA 2020. All rights reserved

Parámetros

- Las canalizaciones admiten su personalización aceptando parámetros especificados por el usuario en tiempo de ejecución. La configuración de parámetros se realiza a través de la directiva parameters. Si se configura la canalización para aceptar parámetros mediante la opción Generar con parámetros, se puede acceder a esos parámetros como miembros de la variable params.

```
pipeline {  
    agent any  
    parameters {  
        string(name: 'Greeting', defaultValue: 'Hello', description: 'How should I greet the world?')  
    }  
    stages {  
        stage('Example') {  
            steps {  
                echo "${params.Greeting} World!"  
            }  
        }  
    }  
}
```

© JMA 2020. All rights reserved

Opciones

- La directiva options permite configurar opciones específicas de la canalización desde dentro del propio Pipeline.
- Pipeline proporciona varias de estas opciones, como buildDiscarder, pero también pueden ser proporcionadas por complementos, como timestamps.
- La directiva options es opcional pero solo puede aparecer una vez en la raíz del Pipeline.

```
pipeline {  
    agent any  
    options {  
        retry(3)  
        timeout(time: 1, unit: 'HOURS')  
    }  
    stages {  
        ...  
    }  
}
```
- Cada etapa puede contar con una directiva options pero solo puede contener pasos como retry, timeout, timestamps u opciones declarativas que sean relevantes para un stage, como skipDefaultCheckout.

© JMA 2020. All rights reserved

Etapas y Pasos

- Una canalización de entrega continua es una expresión automatizada del proceso para obtener software desde el control de versiones hasta el despliegue.
- Las canalizaciones se dividen en etapas que se componen de varios pasos que nos permiten crear, probar e implementar aplicaciones. Jenkins Pipeline nos permite componer múltiples pasos de una manera fácil que pueden ayudar a modelar cualquier tipo de proceso de automatización.
- Hay que pensar en un "paso" como un solo comando que realiza una sola acción. Cuando un paso tiene éxito, pasa al siguiente paso. Cuando un paso no se ejecuta correctamente, la canalización fallará. Cuando todos los pasos de la canalización se han completado con éxito, se considera que la canalización se ha ejecutado con éxito.
- La sección stages contiene una o más directivas stage. Cada stage debe tener un nombre, una sección steps (con los pasos de la etapa) o stages (etapas anidadas) y, opcionalmente, secciones agent, post,, ...

© JMA 2020. All rights reserved

Generador de fragmentos

- La utilidad integrada "Snippet Generator" es útil para crear fragmentos de código para pasos individuales, descubrir nuevos pasos proporcionados por complementos o experimentar con diferentes parámetros para un paso en particular y acceder a la documentación.
- El generador de fragmentos se completa dinámicamente con una lista de los pasos disponibles para la instancia de Jenkins. La cantidad de pasos disponibles depende de los complementos instalados que exponen explícitamente los pasos para su uso en Pipeline. La mayoría de los parámetros son opcionales y se pueden omitir en su secuencia de comandos, dejándolos en los valores predeterminados.
- Para generar un fragmento de paso con el Generador de fragmentos:
 1. Navegar hasta el vínculo Sintaxis de Pipeline (Pipeline Syntax) desde la configuración del proyecto Pipeline o en <http://localhost:50080/pipeline-syntax/>.
 2. Seleccionar el paso deseado en el menú desplegable Paso de muestra
 3. Usar el área generada dinámicamente debajo del menú desplegable Paso de muestra para configurar el paso seleccionado.
 4. Hacer clic en Generar script de canalización para crear un fragmento de canalización que se puede copiar y pegar en una canalización.

© JMA 2020. All rights reserved

Pasos básicos

- sh: Ejecutar Shell Script
- bat: Ejecutar Windows Batch Script
- powershell: Ejecutar Windows PowerShell Script
- pwsh: Ejecutar PowerShell Core Script
- echo: Escribir en la salida de la consola
- mail: Enviar un correo electrónico
- tool: Utilizar una herramienta de una instalación de herramienta predefinida
- error: Lanzar error
- catchError: Capturar el error y establecer el resultado de compilación en FAILURE
- warnError: Detectar el error y configurar el resultado de compilación y etapa como UNSTABLE
- unstable: Establecer el resultado de la etapa en UNSTABLE
- git: Realizar una clonación desde el repositorio especificado.
- checkout scm: extraer el código del control de versiones vinculado

© JMA 2020. All rights reserved

Pasos básicos

- isUnix: Comprueba si se ejecuta en un nodo similar a Unix
- pwd: Determinar el directorio actual
- dir: Cambiar el directorio actual
- deleteDir: Eliminar recursivamente el directorio actual del espacio de trabajo
- fileExists: Verificar si el archivo existe en el espacio de trabajo
- readFile: Leer archivo desde el espacio de trabajo
- writeFile: Escribir archivo en el espacio de trabajo
- stash: Guardar algunos archivos para usarlos más adelante en la compilación
- unstash: Restaurar archivos previamente escondidos
- archive: Archivar artefactos
- unarchive: Copiar artefactos archivados en el espacio de trabajo
- withEnv: Establecer variables de entorno
- withContext: use un objeto contextual de las API internas dentro de un bloque
- getContext: obtener objeto contextual de las API internas

© JMA 2020. All rights reserved

Tiempos de espera y reintentos

- Hay algunos pasos especiales que "envuelven" a otros pasos y que pueden resolver fácilmente problemas como reintentar (retry) los pasos hasta que tengan éxito o salir si un paso dura demasiado tiempo (timeout). Cuando no se puede completar un paso, los tiempos de espera ayudan al controlador a evitar el desperdicio de recursos. Podemos anidar un retry en un timeout para que los reintentos no excedan un tiempo máximo. Si vence el timeout falla la etapa.

```
steps {
    retry(3) {
        sh './remoteload.sh'
    }
    timeout(time: 3, unit: 'MINUTES') {
        sh './health-check.sh'
    }
}
```
- Con el paso sleep se pausa la construcción hasta que haya expirado la cantidad de tiempo dada.
- El paso waitUntil recorre su cuerpo repetidamente hasta que obtiene true. Si obtiene false, espera un rato y vuelve a intentarlo. No hay límite para el número de reintentos, pero si el cuerpo arroja un error, se propaga inmediatamente.

© JMA 2020. All rights reserved

Terminando

- Cuando la canalización haya terminado de ejecutarse, es posible que deba ejecutar pasos de limpieza o realizar algunas acciones según el resultado del Pipeline. Estas acciones se pueden realizar en la sección post del pipeline o de la etapa.

```
post {  
    always {  
        echo 'Esto siempre se ejecutará'  
    }  
    success {  
        echo 'Esto se ejecutará solo si tiene éxito'  
    }  
    failure {  
        echo 'Esto se ejecutará solo si falla'  
    }  
    unstable {  
        echo 'Esto se ejecutará solo si la ejecución se marcó como inestable'  
    }  
}
```

© JMA 2020. All rights reserved

Terminando

- La sección post define uno o más pasos adicionales que se ejecutan al finalizar la ejecución de una canalización o etapa. Admite bloques de condiciones que permiten la ejecución de pasos dentro de cada condición dependiendo del estado de finalización del Pipeline o etapa. Los bloques de condición se ejecutan en el siguiente orden:
 - always: siempre, independientemente del estado de finalización de la ejecución de Pipeline o etapa.
 - changed: cuando tiene un estado de finalización diferente al de su ejecución anterior.
 - fixed: cuando es exitosa y la ejecución anterior falló o era inestable.
 - regression: cuando falla, es inestable o se cancela y la ejecución anterior fue exitosa.
 - aborted: cuando se cancela, generalmente debido a que la canalización se anuló manualmente.
 - failure: cuando falla.
 - success: cuando es exitosa.
 - unstable: cuando es "inestable", generalmente causado por pruebas fallidas, errores de código, ...
 - unsuccessful: cuando es no es exitosa.
 - cleanup: cuando termine con las anteriores, independientemente del estado.

© JMA 2020. All rights reserved

Notificaciones

- Dado que se garantiza que la sección post de un Pipeline se ejecutará al final de la ejecución de una canalización, podemos agregar alguna notificación u otras tareas de finales en función de como termine el Pipeline.

- Hay muchas formas de enviar notificaciones sobre un Pipeline como enviar notificaciones a un correo electrónico, una sala de Hipchat o un canal de Slack.

```
post {  
    failure {  
        mail to: 'team@example.com',  
            subject: "Failed Pipeline: ${currentBuild.displayName}",  
            body: "Something is wrong with ${env.BUILD_URL}"  
    }  
}
```

- El paso mail acepta: *subject, body, to, cc, bcc, replyTo, from, charset, mimeType*.

© JMA 2020. All rights reserved

Grabación de pruebas

- Las pruebas son una parte fundamental de una buena canalización de entrega continua.

- Jenkins puede registrar y agregar los resultados de las pruebas siempre que el test runner pueda generar archivos de resultados de pruebas.

```
stage("Unit test") {  
    steps {  
        sh "mvn test"  
    }  
    post {  
        always {  
            junit 'target/surefire-reports/*.xml'  
        }  
    }  
}
```

- Jenkins generalmente incluye el paso junit, pero si el test runner no puede generar informes XML de estilo JUnit, existen complementos adicionales que procesan prácticamente cualquier formato de informe de prueba ampliamente utilizado.

© JMA 2020. All rights reserved

Etapasopcionales

- La directiva when permite que Pipeline determine si la etapa debe ejecutarse según la condición dada. Debe contener al menos una condición, si contiene más de una condición, todas las condiciones deben devolver verdadero para que se ejecute la etapa (allOf). Si se usa una condición anyOf, se omiten las pruebas restantes tan pronto como se encuentra la primera condición "verdadera". Se pueden construir estructuras condicionales más complejas utilizando las condiciones de anidamiento: not, allOf o anyOf, y se pueden anidar a cualquier profundidad arbitraria.
- En la condición pueden participar: branch, buildingTag, changelog, changeset, changeRequest, environment, equals, expression, tag, triggeredBy.

```
stage('Deploy') {
    when {
        branch 'production'
        expression { currentBuild.result == null || currentBuild.result == 'SUCCESS' }
        anyOf {
            environment name: 'DEPLOY_TO', value: 'production'
            environment name: 'DEPLOY_TO', value: 'staging'
        }
    }
} steps {
```

© JMA 2020. All rights reserved

Paralelismo

- Las etapas en la canalización pueden tener una sección parallel que contenga una lista de etapas anidadas que se ejecutarán en paralelo. Una etapa solo debe una sección steps, stages, parallel o matrix, que son excluyentes.
- Se puede forzar la cancelación de todas las etapas paralelas cuando cualquiera de ellas falla, agregando failFast true al stage que contiene la sección parallel.

```
stage('Parallel Stage') {
    failFast true
    parallel {
        stage('Branch A') {
            agent { label "for-branch-a" }
            steps {
                // ...
            }
        }
        stage('Branch B') {
            agent { label "for-branch-b" }
        }
    }
}
```

© JMA 2020. All rights reserved

Matrices: múltiples combinaciones

- Las etapas pueden tener una sección matrix que defina una matriz multidimensional de combinaciones de nombre y valor para ejecutarse en paralelo. Nos referiremos a estas combinaciones como "celdas" en una matriz. Cada celda en una matriz puede incluir una o más etapas para ejecutarse secuencialmente usando la configuración para esa celda. Se puede forzar la cancelación del resto de celdas cuando cualquiera de ellas falla, agregando failFast true.
- La sección matrix debe incluir una sección axes y una sección stages. La sección axes define los valores para cada uno ejes en la matriz. La sección stages define las etapas a ejecutar secuencialmente en cada celda. La matriz puede tener una sección excludes para eliminar celdas no válidas de la matriz.

```
matrix {  
    axes {  
        axis { name 'PLATFORM' values 'linux', 'mac', 'windows' }  
        axis { name 'BROWSER' values 'chrome', 'edge', 'firefox', 'safari' }  
        axis { name 'ARCHITECTURE' values '32-bit', '64-bit' }  
    }  
    excludes {  
        exclude {  
            axis { name 'PLATFORM' values 'mac' }  
            axis { name 'ARCHITECTURE' values '32-bit' }  
        }  
        exclude {  
            axis { name 'PLATFORM' values 'linux' }  
            axis { name 'BROWSER' values 'safari' }  
        }  
    }  
    failFast true  
    stages {  
        // ...  
    }  
}
```

© JMA 2020. All rights reserved

Despliegue

- La canalización de entrega continua más básica tendrá, como mínimo, tres etapas que deben definirse en un Jenkinsfile: construcción, prueba e implementación. Las etapas estables de construcción y prueba son un precursor importante de cualquier actividad de despliegue.
- Un patrón común es ampliar la cantidad de etapas para capturar entornos de implementación adicionales, como "pre producción" o "producción":

```
stage('Deploy - Staging') {  
    steps {  
        sh './deploy staging'  
        sh './run-smoke-tests'  
        sh './run-acceptance-tests'  }  
}  
stage('Deploy - Production') {  
    steps {  
        sh './deploy production'  
    }  
}
```

© JMA 2020. All rights reserved

Intervención humana

- La canalización que implementa automáticamente el código hasta la producción puede considerarse una implementación de "despliegue continuo". Si bien este es un ideal noble, para muchos hay buenas razones por las que el despliegue continuo puede no ser práctico, pero aún pueden disfrutar de los beneficios de la entrega continua. Jenkins Pipeline es compatible con ambos.
- A menudo, al pasar entre etapas, especialmente cuando cambia el entorno, es posible que se desee la participación humana antes de continuar. Por ejemplo, para juzgar si la aplicación está en un estado lo suficientemente bueno como para "promocionar" al entorno de producción. Esto se puede lograr con el paso input: bloquea la entrada y no continuará sin que una persona confirme el progreso.

```
stage('Deliver') {
    steps {
        sh './jenkins/scripts/deliver.sh'
        input message: 'Finished using the web site? (Click "Proceed" to continue)'
    }
}
```

© JMA 2020. All rights reserved

Docker con Pipeline

- Muchas organizaciones usan Docker para unificar sus entornos de compilación y prueba en todas las máquinas y para proporcionar un mecanismo eficiente para implementar aplicaciones. Las versiones 2.5 y posteriores de Pipeline tienen soporte integrado para interactuar con Docker desde dentro del Jenkinsfile.
- Pipeline está diseñado para usar fácilmente imágenes de Docker como entorno de ejecución para una sola etapa o toda la canalización. Lo que significa que un usuario puede definir las herramientas requeridas para su Pipeline, sin tener que configurar agentes manualmente. Prácticamente cualquier herramienta que se pueda empaquetar en un contenedor Docker se puede usar con facilidad haciendo solo modificaciones menores en un archivo Jenkinsfile.

```
pipeline {
    agent {
        docker { image 'node:16.13.1-alpine' }
    }
}
```

© JMA 2020. All rights reserved

multiple-containers

```
pipeline {
    agent none
    stages {
        stage("Back-end") {
            agent {
                docker { image 'maven:3.8.1-adoptopenjdk-11' }
            }
            steps {
                sh 'mvn --version'
            }
        }
        stage("Front-end") {
            agent {
                docker { image 'node:16.13.1-alpine' }
            }
            steps {
                sh 'node --version'
            }
        }
    }
}
```

© JMA 2020. All rights reserved

node-react

```
pipeline {
    agent {
        docker {
            image 'node:ts-buster-slim'
            args '-p 3000:3000'
        }
    }
    environment {
        CI = 'true'
    }
    stages {
        stage ('Clone') {
            steps {
                git url: 'https://github.com/jenkins-docs/simple-node-js-react-npm-app'
            }
        }
        stage('Build') {
            steps {
                sh 'npm install'
            }
        }
        stage('Test') {
            steps {
                sh './jenkins/scripts/test.sh'
            }
        }
        stage('Deliver') {
            steps {
                sh './jenkins/scripts/deliver.sh'
                input message: 'Finished using the web site? (Click "Proceed" to continue)'
                sh './jenkins/scripts/kill.sh'
            }
        }
    }
}
```

© JMA 2020. All rights reserved

Almacenamiento en caché de datos para contenedores

- Muchas herramientas de compilación descargarán dependencias externas y las almacenarán en caché localmente para reutilizarlas en el futuro. Dado que los contenedores se crean inicialmente con sistemas de archivos "limpios", esto puede generar Pipelines más lentos, ya que es posible que no aprovechen las cachés en disco entre ejecuciones posteriores de Pipeline.
- Pipeline admite la adición de argumentos personalizados que se pasan a Docker, lo que permite a los usuarios especificar volúmenes de Docker personalizados para montar, que se pueden usar para almacenar datos en caché en el agente entre ejecuciones de Pipeline.

```
pipeline {  
    agent {  
        docker {  
            image 'maven:3.8.1-adoptopenjdk-11'  
            args '-v $HOME/.m2:/root/.m2'  
        }  
    }  
}
```

© JMA 2020. All rights reserved

Usando un Dockerfile

- Para proyectos que requieren un entorno de ejecución más personalizado, Pipeline también admite la creación y ejecución de un contenedor desde un Dockerfile del repositorio de origen. En contraste con el enfoque anterior de usar un contenedor "listo para usar", usar la sintaxis `agent { dockerfile true }` creará una nueva imagen a partir de un Dockerfile en lugar de extraer una de Docker Hub .

```
FROM node:16.13.1-alpine  
RUN apk add -U subversion
```

- Al enviar esto a la raíz del repositorio de origen, Jenkinsfile se puede cambiar para crear un contenedor basado en este Dockerfile y luego ejecutar los pasos definidos usando ese contenedor:

```
pipeline {  
    agent { dockerfile true }
```

© JMA 2020. All rights reserved

Continuous Deployment: NodeJS

```
pipeline {
    agent any
    environment {
        REGISTRY = 'jamarton/mock-web-server'
    }
    stages {
        stage('Checkout') {
            steps {
                // Get Github repo using Github credentials (previously added to Jenkins credentials)
                git url: 'https://github.com/jmagit/MOCKWebServer'
            }
        }
        stage('Install dependencies') {
            steps {
                sh 'npm --version'
                sh "npm install"
            }
        }
        stage('Unit tests') {
            steps {
                // echo 'Run unit tests'
                sh "npm run test"
            }
        }
        stage("SonarQube Analysis") {
            environment {
                // Previously defined in the Jenkins "Global Tool Configuration"
                scannerHome = tool 'SonarQube Scanner'
            }
            steps {
                withSonarQubeEnv('SonarQubeDockerServer') {
                    sh "${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=MOCKWebServer \
                        -Dsonar.sources=./src \
                        -Dsonar.tests=./spec \
                        -Dsonar.javascript.lcov.reportPaths=/coverage/lcov.info"
                }
                timeout(5) // time: 5 unit: 'MINUTES'
                // In case of SonarQube failure or direct timeout exceed, stop Pipeline
                waitForQualityGate abortPipeline:waitForQualityGate().status != 'OK'
            }
        }
        stage('Build docker-image') {
            steps {
                sh "docker build -t ${REGISTRY}:${BUILD_NUMBER} ."
            }
        }
        stage('Deploy docker-image') {
            steps {
                // If the Dockerhub authentication stopped, do it again
                // sh 'docker login'
                // sh "docker push ${REGISTRY}:${BUILD_NUMBER}"
                echo 'docker push'
            }
        }
    }
}
```

© JMA 2020. All rights reserved

Blue Ocean (obsoleto)

- Blue Ocean en su forma actual proporciona una visualización de Pipeline fácil de usar. Estaba destinado a ser un replanteamiento de la experiencia del usuario de Jenkins, diseñado desde cero para Jenkins Pipeline. Blue Ocean estaba destinado a reducir el desorden y aumentar la claridad para todos los usuarios.
- Las principales características de Blue Ocean incluyen:
 - Visualización sofisticada de Pipelines de entrega continua (CD), lo que permite una comprensión rápida e intuitiva del estado de su Pipeline.
 - El editor de Pipeline hace que la creación de Pipelines sea más accesible al guiar al usuario a través de un proceso visual para crear un Pipeline.
 - Personalización para adaptarse a las necesidades basadas en roles de cada miembro del equipo.
 - Determinar con precisión para cuando se necesita intervención o surgen problemas. Blue Ocean muestra dónde se necesita atención, lo que facilita el manejo de excepciones y aumenta la productividad.
 - Integración nativa para sucursales y solicitudes de incorporación de cambios, lo que permite la máxima productividad del desarrollador al colaborar en código en GitHub y Bitbucket.
- *Blue Ocean no recibirá más funciones ni actualizaciones de mejoras.*

© JMA 2020. All rights reserved

Blue Ocean (obsoleto)

✓ demos-devops < 11

Rama: main ⚡ 1m 20s Modificado por Javier Martín
Commit: e8b7dfa ⚡ 3 days ago Ejecutado porque se detectaron cambios en el repositorio

Pipeline Modificación Pruebas Artefacto ⚡ 🖊️⚙️🔗Disconnect X

```
graph LR; Start((Start)) --> Initialize((Initialize)); Initialize --> Compile((Compile)); Compile --> UnitTest((Unit test)); UnitTest --> SonarQubeAnalysis((SonarQube Analysis)); SonarQubeAnalysis --> Build((Build)); Build --> Deploy((Deploy)); Deploy --> End((End));
```

SonarQube Analysis - 34s

Restart SonarQube Analysis ⚡ ↴

- ✓ > mvn clean verify sonarsonar -- Shell Script 30s
- ✓ ▾ Wait for SonarQube analysis to be completed and return quality gate status 3s
 - 1 Checking status of SonarQube task 'AYSwd2cfvX0DwWe_4yhx' on server 'SonarQubeDockerServer'
 - 2 SonarQube task 'AYSwd2cfvX0DwWe_4yhx' status is 'PENDING'
 - 3 SonarQube task 'AYSwd2cfvX0DwWe_4yhx' status is 'SUCCESS'
 - 4 SonarQube task 'AYSwd2cfvX0DwWe_4yhx' completed. Quality gate is 'OK'
- ✓ ▾ false -- Wait for SonarQube analysis to be completed and return quality gate status <1s
 - 1 Checking status of SonarQube task 'AYSwd2cfvX0DwWe_4yhx' on server 'SonarQubeDockerServer'
 - 2 SonarQube task 'AYSwd2cfvX0DwWe_4yhx' status is 'SUCCESS'
 - 3 SonarQube task 'AYSwd2cfvX0DwWe_4yhx' completed. Quality gate is 'OK'

© JMA 2020. All rights reserved