Documento técnico de AWS

Práctica de integración y entrega continuas en AWS

Práctica de integración y entrega continuas en AWS: Documento técnico de AWS

Table of Contents

Práctica de integración y entrega continuas en AWS

Fecha de publicación: 27 de octubre de 2021 (Revisiones del documento)

Resumen

En este documento se explican las características y los beneficios de utilizar la integración y la entrega continuas (CI/CD) junto con las herramientas de Amazon Web Services (AWS) en su entorno de desarrollo de software. La integración continua y la entrega continua son prácticas recomendadas y una parte vital de una iniciativa de DevOps.

El desafío de la entrega de software

Las empresas de hoy enfrentan los desafíos de los entornos competitivos que cambian rápidamente, los requisitos de seguridad en evolución y la escalabilidad del rendimiento. Las empresas deben cerrar la brecha entre la estabilidad de las operaciones y el rápido desarrollo de características. La integración continua y la entrega continua (CI/CD) son prácticas que permiten cambios rápidos de software a la vez que mantienen la estabilidad y la seguridad del sistema.

Amazon se dio cuenta desde el principio de que las necesidades empresariales de ofrecer características a los clientes minoristas de Amazon.com, las sucursales de Amazon y Amazon Web Services (AWS) requerirían formas nuevas e innovadoras de entregar software. A la escala de una empresa como Amazon, miles de equipos de software independientes deben poder trabajar en paralelo para entregar software de forma rápida, segura, fiable y sin tolerancia a las interrupciones.

Al aprender a ofrecer software a alta velocidad, Amazon y otras organizaciones con visión de futuro fueron pioneras en DevOps. DevOps es una combinación de herramientas, prácticas y filosofías culturales que aumenta la capacidad de una organización para entregar aplicaciones y servicios a mayor velocidad. Con los principios de DevOps, las organizaciones pueden evolucionar y mejorar los productos a un ritmo más rápido que las organizaciones que utilizan procesos tradicionales de desarrollo de software y administración de infraestructura. Esta velocidad permite a las organizaciones proporcionar un mejor servicio a sus clientes y competir de forma más eficaz en el mercado.

Algunos de estos principios, como los equipos reducidos y la arquitectura orientada a microservicios/servicios (SOA), están fuera del alcance de este documento técnico. En este documento técnico se analiza la capacidad de CI/CD que Amazon ha creado y mejorado continuamente. La CI/CD es clave para ofrecer características de software de manera rápida y fiable.

AWS ahora ofrece estas capacidades de CI/CD como un conjunto de servicios para desarrolladores: AWS CodeStar, AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy y AWS CodeArtifact. Los desarrolladores y los profesionales de operaciones de TI que practican DevOps pueden usar estos servicios para entregar software de manera rápida y segura. En conjunto, le ayudan a almacenar y aplicar al código fuente de su aplicación el control de versiones de forma segura. Puede utilizar AWS CodeStar para orquestar rápidamente un flujo de trabajo de lanzamiento de software integral mediante estos servicios. Para un entorno existente,AWS CodePipeline tiene la flexibilidad de integrar cada servicio de forma independiente con sus herramientas existentes. Se trata de servicios de alta disponibilidad y fácil integración a los que se puede acceder a través de la

AWS Management Console, interfaces de programación de aplicaciones (API) de AWS y los kits de herramientas de desarrollo de software (SDK) de AWS, como cualquier otro servicio de AWS.

¿Qué es la integración continua y la entrega/implementación continua?

En esta sección se analizan las prácticas de integración continua y entrega continua y se explica la diferencia entre la entrega continua y la implementación continua.

Integración continua

La integración continua (CI) es un método de desarrollo de software en el que los desarrolladores fusionan periódicamente sus cambios de código en un repositorio central y, con posterioridad, se ejecutan compilaciones y pruebas automatizadas. CI hace referencia comunmente a la etapa de compilación o integración del proceso de lanzamiento de software y que requiera tanto un componente de automatización (por ejemplo un servicio de compilación o integración continua) como un componente cultural (por ejemplo aprender a realizar integraciones frecuentes). Los objetivos clave de CI consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y lanzar nuevas actualizaciones de software.

La integración continua se centra en confirmaciones más pequeñas y cambios de código más pequeños para su integración. Un desarrollador confirma código a intervalos regulares, como mínimo una vez al día. El desarrollador extrae código del repositorio de código para garantizar que el código en el host local se fusione antes de enviarlo al servidor de compilación. En esta etapa, el servidor de compilación ejecuta las distintas pruebas y acepta o rechaza la confirmación del código.

Los desafíos básicos de implementar la CI incluyen confirmaciones más frecuentes en la base de código común, mantener un único repositorio de código fuente, automatizar compilaciones y automatizar las pruebas. Los desafíos adicionales incluyen pruebas en entornos similares a los de producción, lo que proporciona visibilidad del proceso al equipo y permite a los desarrolladores obtener fácilmente cualquier versión de la aplicación.

Entrega e implementación continuas

La entrega continua (CD) es una práctica de desarrollo de software mediante la cual se compilan, prueban y preparan automáticamente los cambios en el código para la salida a producción. Amplía la integración continua mediante la implementación de todos los cambios de código en un entorno de prueba, un entorno de producción o ambos después de completar la etapa de compilación. La entrega continua se puede automatizar por completo con un proceso de flujo de trabajo o

automatizarse parcialmente con pasos manuales en puntos críticos. Cuando se la entrega continua se implementa de manera adecuada, los desarrolladores disponen siempre de un artefacto listo para su implementación que se ha sometido a un proceso de pruebas estandarizado.

Con la implementación continua, las revisiones se implementan en un entorno de producción automáticamente sin la aprobación explícita del desarrollador, con lo que se automatiza todo el proceso de publicación de software. Esto, a su vez, permite un bucle de retroalimentación de los clientes en las primeras etapas del ciclo de vida del producto.

La entrega continua no es una implementación continua

Una idea errónea sobre la entrega continua es que significa que cada cambio comprometido se aplica a la producción inmediatamente después de pasar las pruebas automatizadas. Sin embargo, el punto de entrega continua no es aplicar todos los cambios a la producción de inmediato, sino garantizar que todos los cambios estén listos para entrar en producción.

Antes de implementar un cambio en producción, puede implementar un proceso de decisión para garantizar que la implementación de producción esté autorizada y auditada. Esta decisión la puede tomar una persona y luego ejecutarla con las herramientas.

Con la entrega continua, la decisión de ponerlo en marcha se convierte en una decisión empresarial, no técnica. La validación técnica ocurre en cada confirmación.

Implementar un cambio en la producción no es un evento disruptivo. La implementación no requiere que el equipo técnico deje de trabajar en el siguiente conjunto de cambios y no necesita un plan de proyecto, documentación de traspaso ni un período de mantenimiento. La implementación se convierte en un proceso repetible que se ha llevado a cabo y probado varias veces en entornos de prueba.

Beneficios de la entrega continua

La CD proporciona numerosos beneficios para su equipo de desarrollo de software, como la automatización del proceso, la mejora de la productividad del desarrollador, la mejora de la calidad del código y la entrega de actualizaciones más rápida a sus clientes.

Automatizar el proceso de lanzamiento de software

CD proporciona un método para que su equipo registre el código que se crea, prueba y prepara automáticamente para su lanzamiento a producción, de modo que la entrega de software sea eficiente, resistente, rápida y segura.

Mejorar la productividad de los desarrolladores

Las prácticas de CD ayudan a la productividad de su equipo liberando a los desarrolladores de tareas manuales, desenredando dependencias complejas y centrándose de nuevo en la entrega de nuevas características en el software. En lugar de integrar su código con otras partes de la empresa y gastar para implementar este código en una plataforma, los desarrolladores pueden centrarse en la lógica de codificación que ofrezca las características que necesita.

Mejorar la calidad del código

La CD puede ayudarle a descubrir y abordar errores al principio del proceso de entrega antes de que se conviertan en problemas mayores más adelante. Su equipo puede realizar fácilmente tipos adicionales de pruebas de código porque todo el proceso se ha automatizado. Con la disciplina de realizar más pruebas con mayor frecuencia, los equipos pueden iterar más rápido con comentarios inmediatos sobre el impacto de los cambios. Esto permite a los equipos impulsar códigos de calidad con una alta garantía de estabilidad y seguridad. Los desarrolladores sabrán a través de comentarios inmediatos si el nuevo código funciona y si se introdujeron cambios o errores importantes. Los errores detectados al principio del proceso de desarrollo son los más fáciles de corregir.

Distribuir actualizaciones más rápido

La CD ayuda a su equipo a entregar actualizaciones a los clientes de forma rápida y frecuente. Cuando se implementa la CI/CD, aumenta la velocidad de todo el equipo, incluida la publicación de

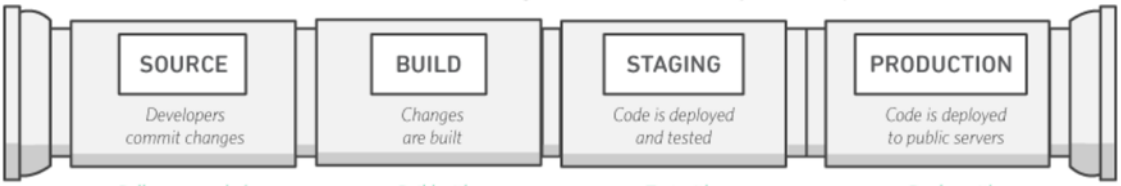
características y la corrección de errores. Las empresas pueden responder más rápido a los cambios del mercado, los desafíos de seguridad, las necesidades de los clientes y las presiones de costes. Por ejemplo, si se requiere una nueva característica de seguridad, su equipo puede implementar CI/CD con pruebas automatizadas para introducir la solución de manera rápida y fiable en los sistemas de producción con alta confianza. Lo que antes tomaba semanas y meses ahora se puede hacer en días o incluso horas.

Implementación de integración continua y entrega continua

En esta sección se describen las formas en que puede empezar a implementar un modelo de CI/CD en su organización. En este documento técnico no se analiza cómo una organización con un modelo maduro de DevOps y transformación de la nube crea y utiliza una canalización de CI/CD. Para ayudarle en su recorrido hacia DevOps, AWS cuenta con varios socios de DevOps certificados que pueden proporcionarle recursos y herramientas. Para obtener más información sobre cómo prepararse para pasar a la nube de AWS, consulte Creación de un modelo operativo en la nube.

Un camino hacia la integración continua/entrega continua

La CI/CD se puede representar como una canalización (consulte la siguiente figura), donde el código nuevo se envía en un extremo, se prueba en una serie de etapas (origen, compilación, ensayo y producción) y, a continuación, se publica como código listo para producción. Si su organización es nueva en CI/CD, puede abordar este proceso de manera iterativa. Esto significa que debe comenzar poco a poco e iterar en cada etapa para poder comprender y desarrollar el código de una manera que ayude a la organización a crecer.



Canalización de CI/CD

Cada etapa de la canalización de CI/CD se estructura como una unidad lógica en el proceso de entrega. Además, etapa actúa como una puerta que examina un cierto aspecto del código. A medida que el código avanza en la canalización, se supone que la calidad del código es mayor en las etapas posteriores porque se siguen verificando más aspectos del mismo. Los problemas descubiertos en una etapa temprana impiden que el código avance a través de la canalización. Los resultados de las pruebas se envían inmediatamente al equipo y todas las compilaciones y versiones posteriores se detienen si el software no pasa la etapa.

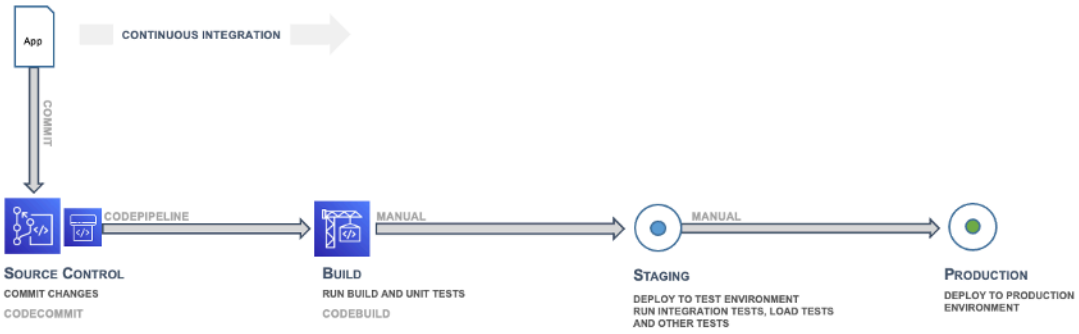
Estas etapas son sugerencias. Puede adaptar las etapas en función de las necesidades de su negocio. Algunas etapas se pueden repetir para varios tipos de pruebas, seguridad y rendimiento. Dependiendo de la complejidad de su proyecto y de la estructura de sus equipos, algunas etapas se pueden repetir varias veces en diferentes niveles. Por ejemplo, el producto final de un equipo puede convertirse en una dependencia en el proyecto del siguiente equipo. Esto significa que el producto

final del primer equipo se presenta posteriormente como un artefacto en el proyecto del siguiente equipo.

La presencia de una canalización de CI/CD tendrá un gran impacto en la maduración de las capacidades de su organización. La organización debe comenzar con pequeños pasos y no intentar crear una canalización completamente madura, con múltiples entornos, muchas fases de prueba y automatización en todas las etapas al principio. Tenga en cuenta que incluso las organizaciones que tienen entornos de CI/CD muy maduros aún necesitan mejorar continuamente sus canalizaciones.

Crear una organización habilitada para CI/CD es un viaje y hay muchos destinos en el camino. La siguiente sección analiza un posible camino que su organización podría tomar, comenzando con la integración continua a través de los niveles de entrega continua.

Integración continua



Integración continua: origen y compilación

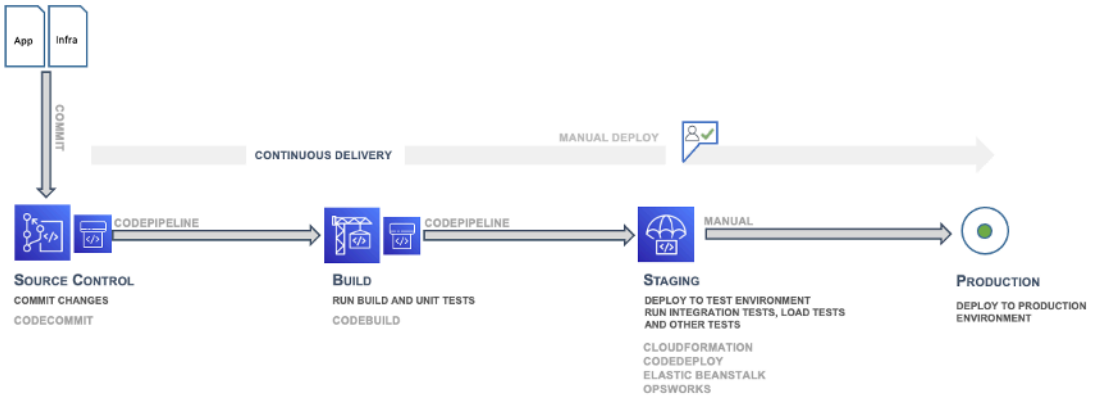
La primera fase en el viaje de CI/CD es desarrollar la madurez en la integración continua. Debe asegurarse de que todos los desarrolladores confirmen regularmente su código en un repositorio central (como uno alojado en CodeCommit o GitHub) y fusionen todos los cambios en una ramificación de lanzamiento para la aplicación. Ningún desarrollador debe mantener el código de forma aislada. Si se necesita una ramificación de función durante un cierto período de tiempo, debe mantenerse actualizada mediante la fusión desde el flujo ascendente con la mayor frecuencia posible. Se recomiendan compromisos y fusiones frecuentes con unidades de trabajo completas para que el equipo desarrolle la disciplina y el proceso los alienta. Un desarrollador que fusione código de forma temprana y frecuente, probablemente tendrá menos problemas de integración en el futuro.

También debe alentar a los desarrolladores a crear pruebas unitarias lo antes posible para sus aplicaciones y a ejecutar estas pruebas antes de enviar el código al repositorio central. Los errores

detectados al principio del proceso de desarrollo de software son los más baratos y fáciles de corregir.

Cuando el código se envía a una rama en un repositorio de código fuente, un motor de flujo de trabajo que monitorea esa ramificación enviará un comando a una herramienta de compilación para compilar el código y ejecutar las pruebas unitarias en un entorno controlado. El proceso de compilación debe tener el tamaño adecuado para manejar todas las actividades, incluidas las presiones y las pruebas que puedan ocurrir durante la etapa de confirmación, para obtener una retroalimentación rápida. En esta etapa también se pueden realizar otras comprobaciones de calidad, como la cobertura de pruebas unitarias, la verificación de estilo y el análisis estático. Por último, la herramienta de compilación crea una o más compilaciones binarias y otros artefactos, como imágenes, hojas de estilo y documentos para la aplicación.

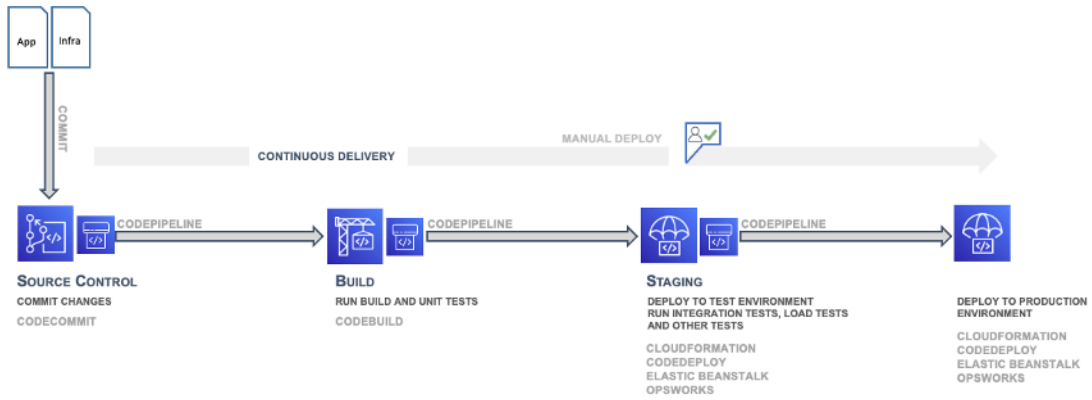
Entrega continua: creación de un entorno de ensayo



Entrega continua: ensayo

La entrega continua (CD) es la siguiente fase e implica implementar el código de la aplicación en un entorno de ensayo, que es una réplica de la pila de producción, y ejecutar más pruebas funcionales. El entorno de ensayo puede ser un entorno estático prediseñado para las pruebas, o puede aprovisionar y configurar un entorno dinámico con una infraestructura y un código de configuración comprometidos para probar e implementar el código de la aplicación.

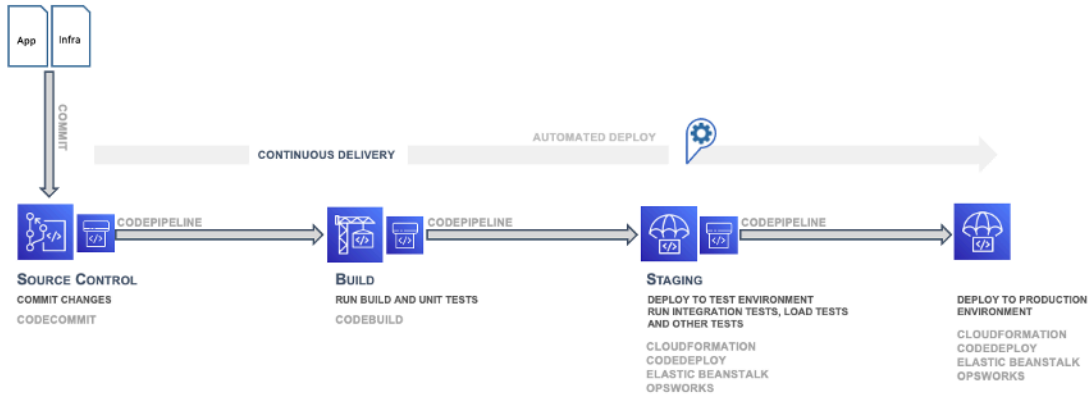
Entrega continua: creación de un entorno de producción



Entrega continua: producción

En la secuencia de canalización de implementación/entrega, después del entorno de ensayo, se encuentra el entorno de producción, que también se construye utilizando la infraestructura como código (IaC).

Implementación continua



Implementación continua

La fase final de la canalización de implementación de CI/CD es la implementación continua, que puede incluir la automatización completa de todo el proceso de lanzamiento de software, incluida la implementación en el entorno de producción. En un entorno de CI/CD completamente maduro, la ruta al entorno de producción está completamente automatizada, lo que permite que el código se implemente con alta confianza.

Madurez y más allá

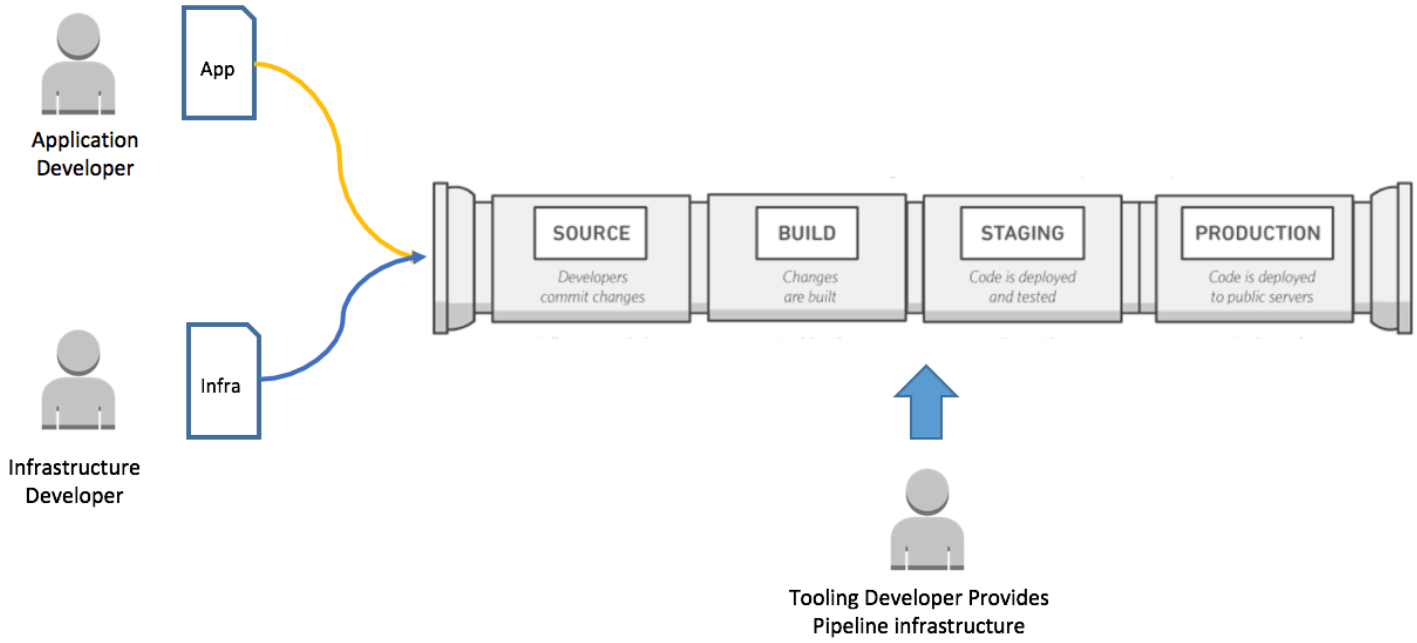
A medida que su organización madure, continuará desarrollando el modelo de CI/CD para incluir más de las siguientes mejoras:

- Más entornos de ensayo para pruebas específicas de rendimiento, cumplimiento, seguridad e interfaz de usuario (IU)
- Pruebas unitarias del código de configuración e infraestructura junto con el código de la aplicación
- Integración con otros sistemas y procesos, como la revisión del código, el seguimiento de problemas y la notificación de eventos
- Integración con la migración de esquemas de base de datos (si procede)
- Pasos adicionales para la auditoría y la aprobación empresarial

Incluso las organizaciones más maduras que tienen canalizaciones complejas de CI/CD multientorno siguen buscando mejoras. DevOps es un viaje, no un destino. Los comentarios sobre la canalización se recopilan de forma continua y se logran mejoras en velocidad, escala, seguridad y fiabilidad en colaboración entre las diferentes partes de los equipos de desarrollo.

Equipos

AWS recomienda organizar tres equipos de desarrolladores para implementar un entorno de CI/CD: un equipo de aplicación, un equipo de infraestructura y un equipo de herramientas (consulte la siguiente figura). Esta organización representa un conjunto de prácticas recomendadas que se han desarrollado y aplicado en empresas emergentes de rápido crecimiento, grandes organizaciones empresariales y en la propia Amazon. Los equipos deben estar compuestos de no más de 10 o 12 personas. Esto sigue una regla de comunicación: las conversaciones significativas alcanzan límites a medida que el tamaño de los grupos aumenta y las líneas de comunicación se multiplican.



Equipos de aplicación, infraestructura y herramientas

Equipo de aplicación

El equipo de aplicación crea la aplicación. Los desarrolladores de la aplicación son propietarios de las tareas pendientes, las historias y las pruebas unitarias, y desarrollan características basadas en un objetivo específico de la aplicación. El objetivo organizativo de este equipo es minimizar el tiempo que estos desarrolladores dedican a tareas no esenciales de la aplicación.

Además de tener habilidades de programación funcionales en el lenguaje de la aplicación, el equipo de la aplicación debe tener habilidades de plataforma y comprender la configuración del sistema. Esto les permitirá centrarse únicamente en desarrollar características y reforzar la aplicación.

Equipo de infraestructura

El equipo de infraestructura escribe el código que crea y configura la infraestructura necesaria para ejecutar la aplicación. Este equipo puede utilizar herramientas AWS nativas como AWS CloudFormation, o herramientas genéricas, como Chef, Puppet o Ansible. El equipo de infraestructura es responsable de especificar qué recursos se necesitan y trabaja en estrecha colaboración con el equipo de aplicación. El equipo de infraestructura puede estar formado por solo una o dos personas para una aplicación pequeña.

El equipo debe tener habilidades en métodos de aprovisionamiento de infraestructura, como AWS CloudFormation o HashiCorp Terraform. El equipo también debe desarrollar habilidades de automatización de la configuración con herramientas como Chef, Ansible, Puppet o Salt.

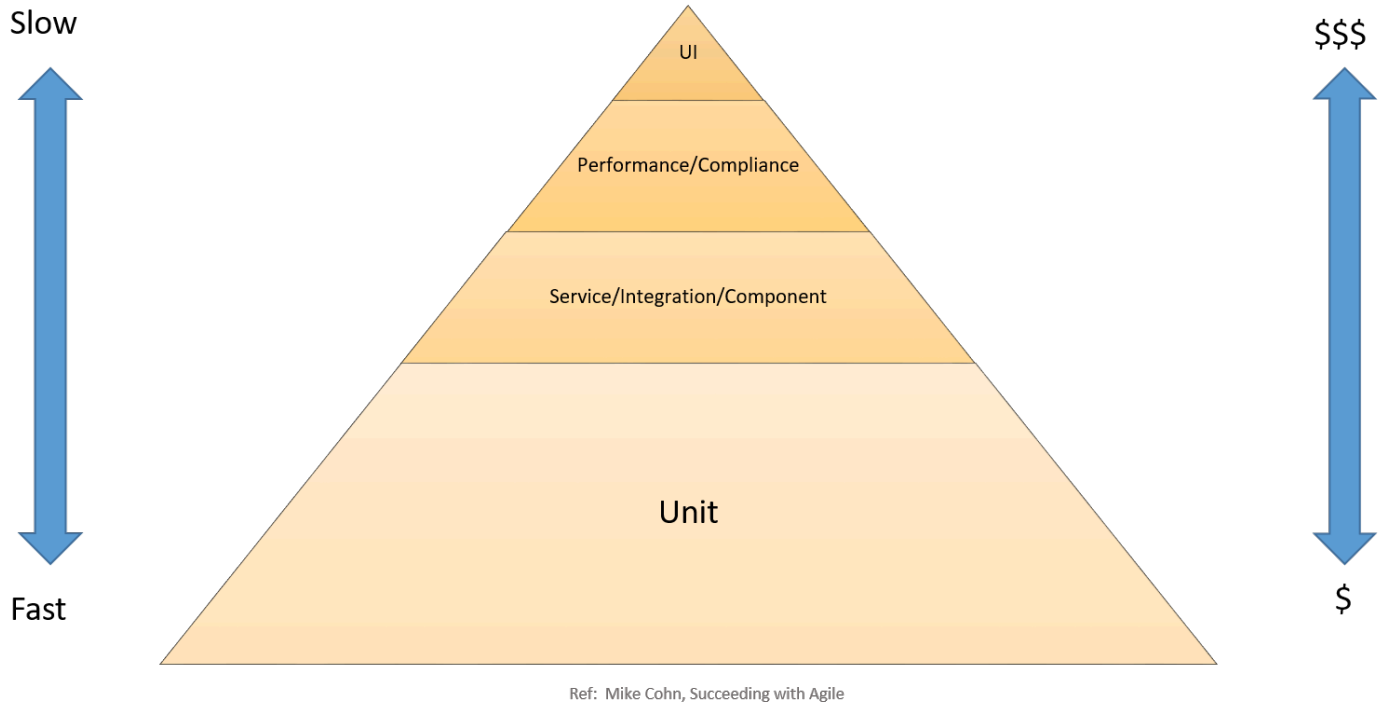
Equipo de herramientas

El equipo de herramientas crea y administra la canalización de CI/CD. ES responsable de la infraestructura y las herramientas que conforman la canalización. No forman parte del equipo de aplicación; sin embargo, crean una herramienta que utilizan los equipos de aplicación e infraestructura en la organización. La organización necesita madurar continuamente su equipo de herramientas, de modo que el equipo de herramientas esté un paso por delante de los equipos de aplicación e infraestructura en proceso de maduración.

El equipo de herramientas debe ser experto en la creación e integración de todas las partes de la canalización de CI/CD. Esto incluye la creación de repositorios de control de código fuente, motores de flujo de trabajo, entornos de compilación, marcos de pruebas y repositorios de artefactos. Este equipo puede optar por implementar software como AWS CodeStar, AWS CodePipeline, AWS CodeCommit, AWS CodeDeploy, AWS CodeBuild, y AWS CodeArtifact, junto con Jenkins, GitHub, Artifactory, TeamCity y otras herramientas similares. Algunas organizaciones pueden llamar a esto un equipo de DevOps, pero AWS lo desaconseja y, en cambio, anima a pensar en DevOps como la suma de las personas, los procesos y las herramientas para la entrega de software.

Etapas de prueba en integración continua y entrega continua

Los tres equipos de CI/CD deben incorporar las pruebas en el ciclo de vida de desarrollo de software en las diferentes etapas de la canalización de CI/CD. En general, las pruebas deben comenzar lo antes posible. La siguiente pirámide de pruebas es un concepto proporcionado por Mike Cohn en Succeeding with Agile. Muestra las diversas pruebas de software en relación con su coste y velocidad a la que se ejecutan.



Pirámide de pruebas de CI/CD

Las pruebas unitarias están en la parte inferior de la pirámide. Son las más rápidas de ejecutar y las menos costosas. Por lo tanto, las pruebas unitarias deben constituir la mayor parte de su estrategia de pruebas. Una buena regla general es alrededor del 70 por ciento. Las pruebas unitarias deben tener una cobertura de código casi completa porque los errores detectados en esta fase se pueden corregir de forma rápida y económica.

Las pruebas de servicio, componentes e integración están en la pirámide por encima de las pruebas unitarias. Estas pruebas requieren entornos detallados y, por lo tanto, son más costosas en cuanto a requisitos de infraestructura y más lentas de ejecutar. Las pruebas de rendimiento y cumplimiento son el siguiente nivel. Requieren entornos de calidad de producción y son aún más caras. Las pruebas de aceptación de la interfaz de usuario y de usuario están en la parte superior de la pirámide y también requieren entornos de calidad de producción.

Todas estas pruebas forman parte de una estrategia completa para garantizar un software de alta calidad. Sin embargo, para acelerar el desarrollo, se hace hincapié en el número de pruebas y la cobertura en la mitad inferior de la pirámide.

En las siguientes secciones se analizan las etapas de la CI/CD.

Configuración del origen

Al principio del proyecto, es esencial configurar un origen en el que poder almacenar el código sin procesar y los cambios de configuración y esquema. En la etapa de origen, elija un repositorio de código fuente, como uno alojado en GitHub o AWS CodeCommit.

Configuración y ejecución de compilaciones

La automatización de compilaciones es esencial para el proceso de CI. Al configurar la automatización de compilaciones, la primera tarea es elegir la herramienta de compilación correcta. Existen muchas herramientas de compilación, tales como:

- Ant, Maven, y Gradle para Java
- Make para C/C++
- Grunt para JavaScript
- Rake para Ruby

La herramienta de compilación que funcione mejor para usted depende del lenguaje de programación de su proyecto y del conjunto de habilidades de su equipo. Después de elegir la herramienta de compilación, todas las dependencias deben definirse claramente en los scripts de compilación, junto con los pasos de compilación. También se recomienda crear una versión de los artefactos de compilación finales, lo que facilita la implementación y el seguimiento de los problemas.

Compilación

En la etapa de compilación, las herramientas de compilación tomarán como entrada cualquier cambio en el repositorio de código fuente, compilarán el software y ejecutarán los siguientes tipos de pruebas:

Pruebas unitarias: prueba una sección específica del código para garantizar que el código hace lo que se espera que haga. Los desarrolladores de software realizan las pruebas unitarias durante la fase de desarrollo. En esta etapa, se puede aplicar un análisis de código estático, un análisis de flujo de datos, una cobertura de código y otros procesos de verificación de software.

Análisis de código estático: esta prueba se realiza sin ejecutar la aplicación después de las pruebas de compilación y unitarias. Este análisis puede ayudar a encontrar errores de codificación y agujeros de seguridad, y también puede garantizar el cumplimiento de las pautas de codificación.

Almacenamiento provisional

En la fase de ensayo, se crean entornos completos que reflejan el entorno de producción final. Se llevan a cabo las siguientes pruebas:

Pruebas de integración: verifica las interfaces entre los componentes en relación con el diseño del software. Las pruebas de integración son un proceso iterativo y facilitan la creación de interfaces robustas y la integridad del sistema.

Pruebas de componentes: prueba el paso de mensajes entre varios componentes y sus resultados. Un objetivo clave de estas pruebas podría ser la idempotencia en las pruebas de componentes. Las pruebas pueden incluir volúmenes de datos extremadamente grandes o situaciones perimetrales y entradas anómalas.

Pruebas del sistema: prueba el sistema de extremo a extremo y verifica si el software cumple con los requisitos comerciales. Esto puede incluir probar la interfaz de usuario (IU), la API, la lógica de backend y el estado final.

Pruebas de rendimiento: determina la capacidad de respuesta y la estabilidad de un sistema a medida que se desempeña bajo una carga de trabajo en particular. Las pruebas de rendimiento también se utilizan para investigar, medir, validar o verificar otros atributos de calidad del sistema, como la escalabilidad, la fiabilidad y el uso de recursos. Los tipos de pruebas de rendimiento pueden incluir pruebas de carga, pruebas de esfuerzo y pruebas de picos. Las pruebas de rendimiento se utilizan para la evaluación comparativa con criterios predefinidos.

Pruebas de cumplimiento: comprueba si el cambio de código cumple con los requisitos de una especificación o reglamento no funcional. Determina si está implementando y cumpliendo con los estándares definidos.

Pruebas de aceptación del usuario: valida el flujo empresarial de extremo a extremo. Esta prueba la ejecuta un usuario final en un entorno de ensayo y confirma si el sistema cumple con los requisitos de la especificación de requisitos. Por lo general, los clientes emplean metodologías de prueba alfa y beta en esta etapa.

Producción

Finalmente, después de pasar las pruebas anteriores, la fase de ensayo se repite en un entorno de producción. En esta fase, se puede completar una prueba de valor controlado final implementando el nuevo código solo en un pequeño subconjunto de servidores o incluso en un servidor, o uno Región

de AWS antes de implementar el código en todo el entorno de producción. Los detalles sobre cómo implementar de forma segura en producción se tratan en la sección Métodos de implementación.

En la siguiente sección se analiza la creación de la canalización para incorporar estas etapas y pruebas.

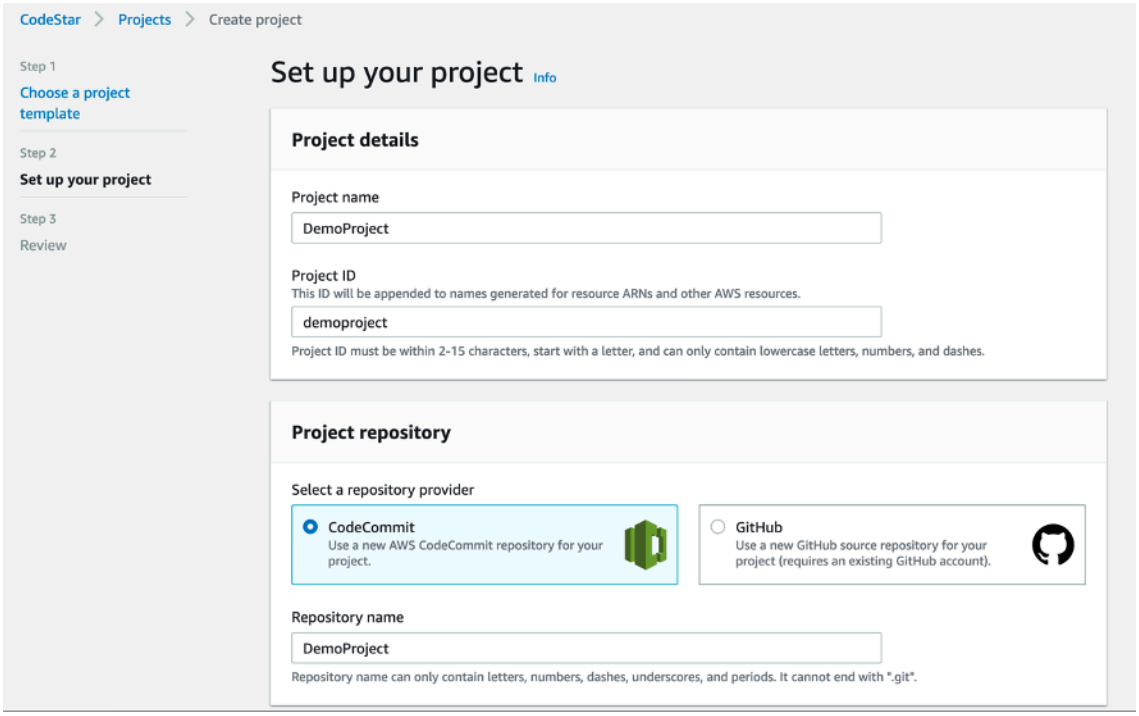
Desarrollo de la canalización

En esta sección se analiza el desarrollo de la canalización. Comience por establecer una canalización con solo los componentes necesarios para la CI y luego haga la transición a una canalización de entrega continua con más componentes y etapas. En esta sección también se explica cómo puede considerar el uso de funciones AWS Lambda y aprobaciones manuales para proyectos grandes, planificar para varios equipos, ramificaciones y Regiones de AWS.

Inicio con una canalización mínima viable para la integración continua

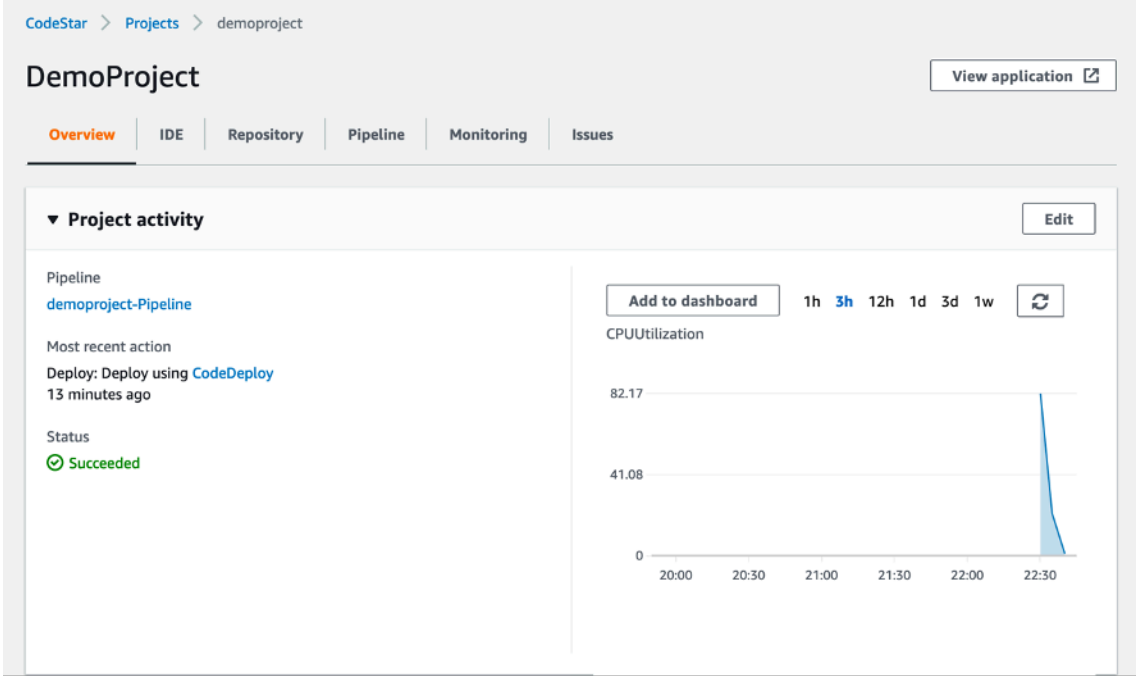
El recorrido de su organización hacia la entrega continua comienza con una canalización mínima viable (MVP). Como se explica en Implementación de la integración continua y la entrega continua, los equipos pueden comenzar con un proceso muy simple, como implementar una canalización que realice una verificación de estilo de código o una prueba de una sola unidad sin implementación.

Un componente clave es una herramienta de orquestación de entrega continua. Para ayudarle a desarrollar esta canalización, Amazon desarrolló AWS CodeStar.



Página de configuración de AWS CodeStar

AWS CodeStar utiliza AWS CodePipeline, AWS CodeBuild AWS CodeCommit y AWS CodeDeploy con un proceso de configuración, herramientas, plantillas y panel integrados. AWS CodeStar proporciona todo lo necesario para desarrollar, compilar e implementar rápidamente aplicaciones en AWS. Esto le permite empezar a publicar código de forma más rápida. Los clientes que ya están familiarizados con la AWS Management Console y buscan un mayor nivel de control pueden configurar manualmente las herramientas de desarrollo que prefieran y pueden proporcionar servicios individuales de AWS según sea necesario.

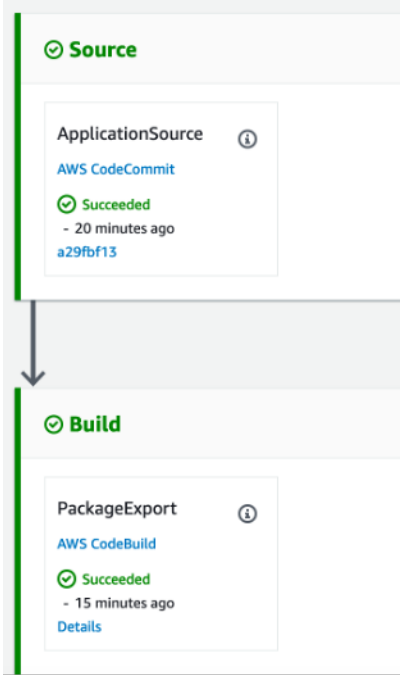


Panel de AWS CodeStar

AWS CodePipeline es un servicio de CI/CD que se puede utilizar a través de AWS CodeStar o a través de la AWS Management Console para obtener actualizaciones rápidas y fiables de aplicaciones e infraestructura. AWS CodePipeline compila, prueba e implementa el código cada vez que hay un cambio de código, en función de los modelos de proceso de lanzamiento que defina. Esto le permite entregar características y actualizaciones de forma rápida y de confianza. Puede desarrollar fácilmente una solución integral usando nuestros complementos prediseñados para servicios de terceros populares como GitHub o integrando sus propios complementos personalizados en cualquier etapa del proceso de lanzamiento. Con AWS CodePipeline, solo paga por lo que usa. No es necesario pagar cuotas iniciales ni asumir compromisos a largo plazo.

Los pasos de AWS CodeStar y AWS CodePipeline se asignan directamente a las etapas de CI/CD de origen, compilación, ensayo y producción. Si bien la entrega continua es deseable, puede

comenzar con una canalización simple de dos pasos que verifique el repositorio de origen y realice una acción de compilación:

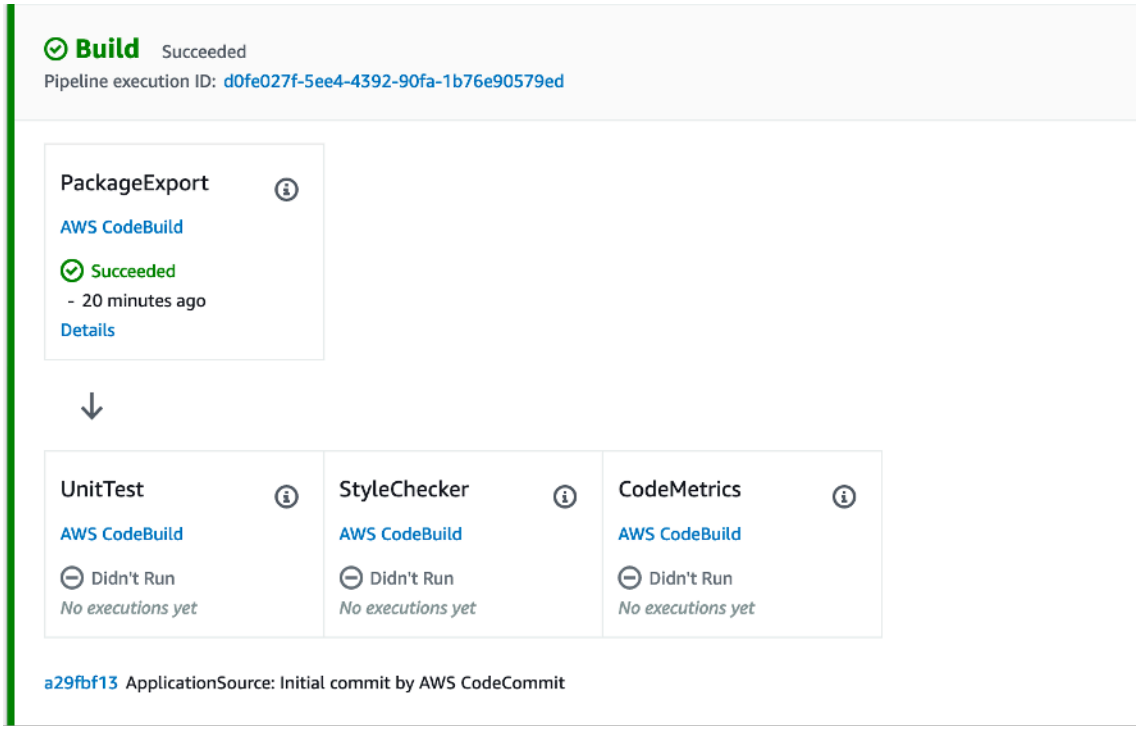


AWS CodePipeline: etapas de origen y desarrollo

Para AWS CodePipeline, la etapa de origen puede aceptar entradas de GitHub y Amazon Simple Storage Service (Amazon S3).AWS CodeCommit La automatización del proceso de desarrollo es un primer paso fundamental para implementar la entrega continua y avanzar hacia la implementación continua. La eliminación de la participación humana en la producción de artefactos de compilación elimina la carga de su equipo, minimiza los errores introducidos por el empaquetado manual y le permite comenzar a empaquetar artefactos consumibles con más frecuencia.

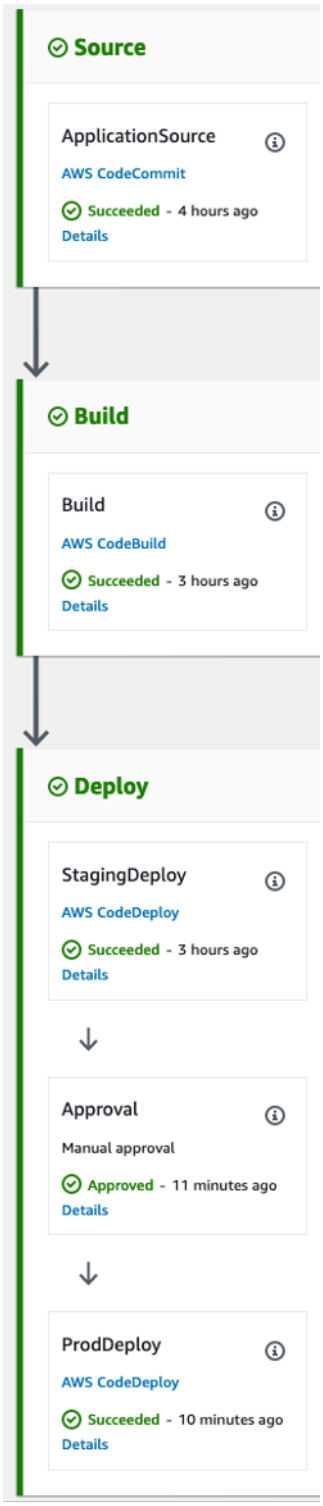
AWS CodePipeline funciona a la perfección con AWS CodeBuild, un servicio de compilación completamente administrada, para facilitar la configuración de un paso de compilación dentro de su canalización que empaquete su código y ejecute pruebas unitarias. Con AWS CodeBuild, no necesita aprovisionar, administrar o escalar sus propios servidores de compilación. AWS CodeBuild se escala continuamente y procesa varias compilaciones al mismo tiempo para que sus compilaciones no se queden esperando en cola. AWS CodePipeline también se integra con servidores de compilación como Jenkins, Solano CI y TeamCity.

Por ejemplo, en la siguiente etapa de compilación, se ejecutan en paralelo tres acciones (pruebas unitarias, verificaciones de estilo de código y recopilación de métricas de código). Al usar AWS CodeBuild, estos pasos se pueden agregar como proyectos nuevos sin ningún esfuerzo adicional en la compilación o instalación de servidores de compilación para manejar la carga.



AWS CodePipeline: funcionalidad de compilación

Las etapas de origen y compilación que se muestran en la figura AWS CodePipeline: etapas de origen y compilación, junto con los procesos de soporte y la automatización, respaldan la transición de su equipo hacia una integración continua. En este nivel de madurez, los desarrolladores deben prestar atención regularmente a los resultados de compilación y prueba. También necesitan crecer y mantener una base de prueba unitaria saludable. Esto, a su vez, refuerza la confianza de todo el equipo en la canalización de CI/CD y promueve su adopción.



Etapas de AWS CodePipeline

Canalización de entrega continua

Una vez que se haya implementado la canalización de integración continua y se hayan establecido los procesos de soporte, sus equipos pueden comenzar la transición hacia la canalización de entrega continua. Esta transición requiere que los equipos automaticen tanto la compilación como la implementación de aplicaciones.

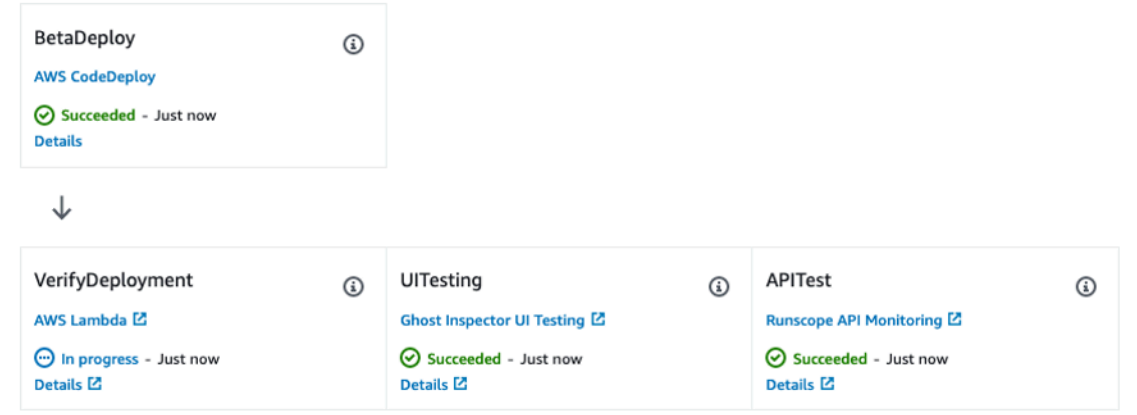
Una canalización de entrega continua se caracteriza por la presencia de etapas de preparación y producción, donde la etapa de producción se realiza después de una aprobación manual.

De la misma manera en que se desarrolló la canalización de integración continua, sus equipos pueden comenzar a crear gradualmente una canalización de entrega continua escribiendo sus scripts de implementación.

Según las necesidades de una aplicación, los servicios de AWS existentes pueden abstraer algunos de los pasos de implementación. Por ejemplo, AWS CodePipeline se integra directamente con AWS CodeDeploy, un servicio que automatiza las implementaciones de código en instancias de Amazon EC2 e instancias que se ejecutan localmente, AWS OpsWorks, un servicio de administración de configuración que le ayuda a operar aplicaciones con Chef y en AWS Elastic Beanstalk, un servicio para implementar y escalar aplicaciones y servicios web.

AWS tiene documentación detallada sobre cómo implementar e integrar AWS CodeDeploy con su infraestructura y canalización.

Una vez que su equipo automatice correctamente la implementación de la aplicación, las etapas de implementación se pueden ampliar con varias pruebas. Por ejemplo, puede añadir otras integraciones listas para usar con servicios como Ghost Inspector, Runscope y otros, como se muestra en la siguiente figura.



AWS CodePipeline: pruebas de código en etapas de implementación

Adición de acciones de Lambda

AWS CodeStar y AWS CodePipeline admiten la integración con AWS Lambda. Esta integración permite implementar un amplio conjunto de tareas, como crear recursos personalizados en su entorno, integrarse con sistemas de terceros (como Slack) y realizar comprobaciones en su entorno recién implementado.

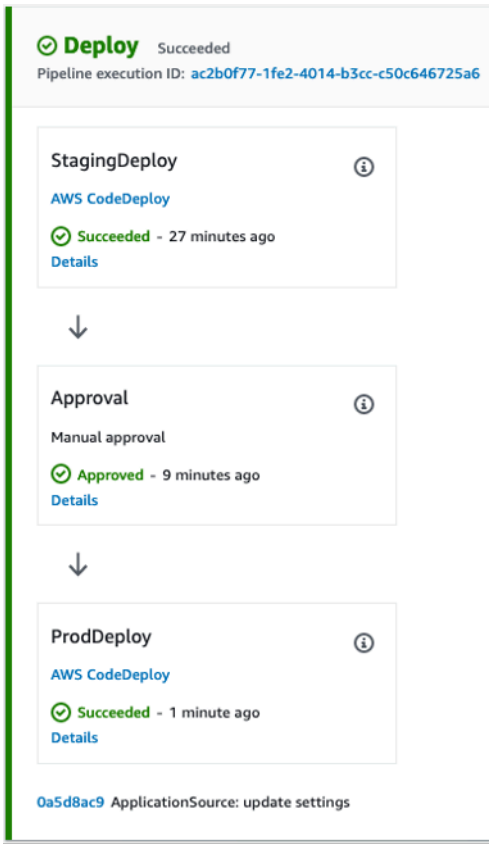
Las funciones de Lambda se pueden usar en canalizaciones de CI/CD para realizar las siguientes tareas:

- Desarrollar cambios en su entorno aplicando o actualizando una plantilla de AWS CloudFormation.
- Crear recursos a demanda en una etapa de la canalización utilizando AWS CloudFormation y eliminarlos en otra.
- Implementar versiones de aplicaciones sin tiempo de inactividad en AWS Elastic Beanstalk con una función de Lambda que intercambia los valores del registro de nombre canónico (CNAME).
- Implementar en instancias Docker de Amazon Elastic Container Service (ECS).
- Hacer una copia de seguridad (es decir, crear un instantánea de AMI) antes de proceder a la creación o la implementación.
- Hacer posible la integración con productos de terceros en su canalización, por ejemplo para publicar mensajes en un cliente Internet Relay Chat (IRC).

Aprobaciones manuales

Añadir una acción de aprobación a una etapa de una canalización en el punto donde quiere que el procesamiento de la canalización se detenga de modo que alguien con los permisos de AWS Identity and Access Management (IAM) necesarios pueda aprobar o rechazar la acción.

Si se aprueba la acción, se reanuda el procesamiento de la canalización. Si la acción se rechaza, o si nadie aprueba o rechaza la acción en el plazo de siete días después de que la canalización alcance la acción y se detenga, el resultado es el mismo que si se produjese un error en la acción y el procesamiento de la canalización no continúa.



AWS CodeDeploy: aprobaciones manuales

Implementación de cambios en el código de infraestructura en una canalización de CI/CD

AWS CodePipeline permite seleccionar AWS CloudFormation como acción de implementación en cualquier etapa de su canalización. A continuación, puede elegir la acción específica que AWS CloudFormation desea realizar, como crear o eliminar pilas y crear o ejecutar conjuntos de cambios. Una pila es un concepto de AWS CloudFormation y representa un grupo de recursos de AWS relacionados. Si bien hay muchas formas de aprovisionar la infraestructura como código, AWS CloudFormation es una herramienta integral recomendada por AWS como una solución escalable y completa que puede describir el conjunto más completo de recursos de AWS como código. AWS recomienda usar AWS CloudFormation en un proyecto de AWS CodePipeline para realizar un seguimiento de los cambios y las pruebas de infraestructura.

CI/CD para aplicaciones sin servidor

También puede utilizar AWS CodeStar,AWS CodePipelineAWS CodeBuild, yAWS CloudFormation para desarrollar canalizaciones de CI/CD para aplicaciones sin servidor. Las aplicaciones sin servidor

integran servicios administrados como Amazon Cognito, Amazon S3 y Amazon DynamoDB con un servicio basado en eventos y AWS Lambda para implementar aplicaciones de una manera que no requiera la administración de servidores. Si es desarrollador de aplicaciones sin servidor, puede utilizar la combinación deAWS CodePipelineAWS CodeBuild, yAWS CloudFormation para automatizar el desarrollo, las pruebas y la implementación de aplicaciones sin servidor que se expresan en plantillas creadas con el AWS Serverless Application Model. Para obtener más información, consulte la documentación de AWS Lambda sobre Automatización de la implementación de aplicaciones basadas en Lambda.

También puede crear canalizaciones de CI/CD seguras que sigan las prácticas recomendadas de su organización con AWS Serverless Application Model Pipelines (AWS SAM Pipelines). Las canalizaciones de AWS SAM son una nueva característica de la CLI de AWS SAM que permite obtener beneficios de CI/CD en cuestión de minutos, como acelerar la frecuencia de las implementaciones, reducir el plazo de entrega de los cambios y disminuir los errores de implementación. AWS SAM Pipelines incluye un conjunto de plantillas de canalización predeterminadas para AWS CodeBuild/CodePipeline que siguen las prácticas recomendadas de implementación de AWS. Para obtener más información y ver el tutorial, consulte el blog Introducing AWS SAM Pipelines.

Canalizaciones para varios equipos, sucursales y regiones de AWS

Para un proyecto grande, no es raro que varios equipos de proyectos trabajen en diferentes componentes. Si varios equipos usan un único repositorio de código, se puede asignar para que cada equipo tenga su propia ramificación. También debe haber una ramificación de integración o lanzamiento para la fusión final del proyecto. Si se utiliza una arquitectura orientada a servicios o de microservicios, cada equipo podría tener su propio repositorio de código.

En el primer escenario, si se usa una sola canalización, es posible que un equipo pueda afectar el progreso de los demás equipos bloqueando la canalización. AWS recomienda crear canalizaciones específicas para las ramificaciones del equipo y otra canalización de lanzamiento para la entrega del producto final.

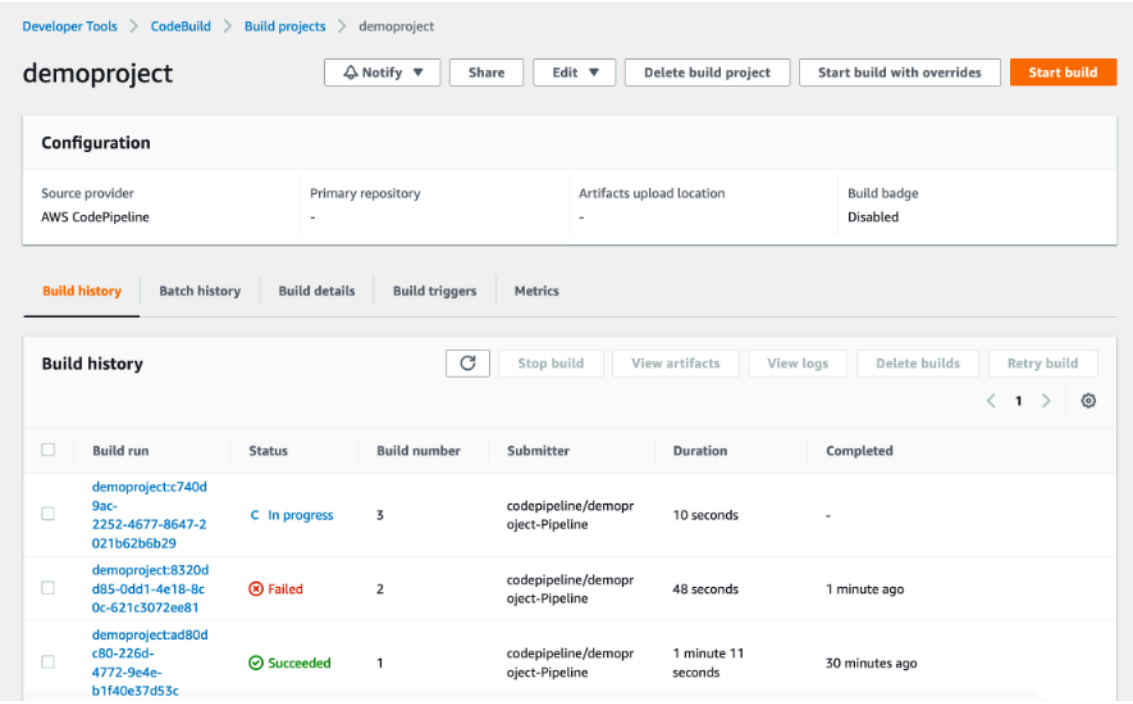
Integración de canalizaciones con AWS CodeBuild

AWS CodeBuild está diseñado para permitir que su organización cree un proceso de creación de alta disponibilidad con una escala casi ilimitada. AWS CodeBuild proporciona entornos de inicio rápido para varios lenguajes populares, además de la capacidad de ejecutar cualquier contenedor de Docker que especifique.

Con las ventajas de una estrecha integración con AWS CodeCommit, AWS CodePipeline y AWS CodeDeploy, así como con las acciones de Git y CodePipeline Lambda, la herramienta CodeBuild es muy flexible.

El software se puede crear mediante la inclusión de un archivo `buildspec.yml` que identifica cada uno de los pasos de compilación, incluidas las acciones previas y posteriores a la compilación, o las acciones especificadas a través de la herramienta CodeBuild.

Puede ver el historial detallado de cada compilación usando el panel de CodeBuild. Los eventos se almacenan como archivos de registro de Amazon CloudWatch Logs.



Archivos de registro de CloudWatch Logs en AWS CodeBuild

Integración de canalizaciones con Jenkins

Puede utilizar la herramienta de compilación de Jenkins para crear canalizaciones de entrega. Estas canalizaciones utilizan trabajos estándar que definen los pasos para implementar etapas de entrega continua. Sin embargo, este enfoque puede que no sea óptimo para proyectos más grandes porque el estado actual de la canalización no persiste entre los reinicios de Jenkins, la implementación de la aprobación manual no es sencilla y el seguimiento del estado de una canalización compleja puede resultar complicado.

En su lugar, AWS recomienda implementar la entrega continua con Jenkins mediante el complemento AWS Code Pipeline. Este complemento permite describir flujos de trabajo complejos utilizando un lenguaje específico de dominio similar a Groovy y se puede usar para orquestar canalizaciones complejas. La funcionalidad del complemento de AWS Code Pipeline se puede mejorar mediante el uso de complementos satelitales, como el complemento Pipeline Stage View, que visualiza el progreso actual de las etapas definidas en una canalización, o el complemento Pipeline Multibranch, que agrupa desarrollos de diferentes ramificaciones.

AWS recomienda que almacene la configuración de su canalización en Jenkinsfile y que la registre en un repositorio de código fuente. Esto permite realizar un seguimiento de los cambios en el código de canalización y se vuelve aún más importante cuando se trabaja con el complemento Pipeline Multibranch. AWStambién recomienda dividir su canalización en etapas. Esto agrupa lógicamente los pasos de la canalización y también permite que el complemento Pipeline Stage View visualice el estado actual de la canalización.

En la siguiente figura se muestra una canalización de ejemplo de Jenkins, con cuatro etapas definidas visualizadas por el complemento Pipeline Stage View.

Etapas definidas de la canalización de Jenkins visualizadas por el complemento Pipeline Stage View

Métodos de implementación

Puede considerar varias estrategias de implementación y variaciones para implementar nuevas versiones de software en un proceso de entrega continuo. En esta sección se describen los métodos de implementación más comunes: todos a la vez (implementación in situ), continua, inmutable y azul/verde. AWS indica cuáles de estos métodos son compatibles con AWS CodeDeploy y AWS Elastic Beanstalk.

En la siguiente tabla se resumen las características de cada método de implementación.

Método	Repercusión de la implementación con errores	Tiempo de implementación	Sin inactividad	Sin cambios de DNS	Proceso de reversión	El código se implementa en
Implementación in situ	Tiempo de inactividad.	🕒	X	✓	Nueva implementación	Instancias existentes
Continua	Solamente un lote se queda fuera de servicio. Todos los lotes implementados correctamente antes del error ejecutan la nueva versión de	🕒 🕒 †	✓	✓	Nueva implementación	Instancias existentes

Método	Repercusión de la implementación con errores	Tiempo de implementación	Sin inactividad	Sin cambios de DNS	Proceso de reversión	El código se implementa en
	la aplicación.					
Continua con un lote adicional (beanstalk)	Mínima si se produce un error en el primer lote; de lo contrario, similar a continua.	🕒 🕒 🕒 †	✓	✓	Nueva implementación	Instancias nuevas y existentes
Inmutable	Mínima	🕒 🕒 🕒 🕒	✓	✓	Nueva implementación	Instancias nuevas
División de tráfico	Mínima	🕒 🕒 🕒 🕒	✓	✓	Redirigir tráfico y finalizar nuevas instancias	Instancias nuevas
Blue/green (azul/verde)	Mínima	🕒 🕒 🕒 🕒	✓	X	volver al entorno anterior	Instancias nuevas

Todo a la vez (implementación in situ)

Todo a la vez (implementación in situ) es un método que puede utilizar para implementar un nuevo código de aplicación en una flota de servidores existente. Este método sustituye todo el código en una acción de implementación. Requiere tiempo de inactividad porque todos los servidores de la flota se actualizan a la vez. No hay necesidad de actualizar los registros de DNS existentes. En caso de un error en la implementación, la única forma de restaurar las operaciones es volver a implementar el código en todos los servidores.

En AWS Elastic Beanstalk, esta implementación se denomina Todo a la vez y está disponible para aplicaciones únicas y con equilibrio de carga. En AWS CodeDeploy, este método de implementación se denomina implementación in situ con una configuración de implementación de `AllAtOnce`.

Implementación continua

Con la implementación continua, la flota se divide en partes para que no se actualice toda la flota a la vez. Durante el proceso de implementación, en la misma flota se ejecutan dos versiones de software, nueva y antigua. Este método permite una actualización sin tiempo de inactividad. Si la implementación falla, solo se verá afectada la parte actualizada de la flota.

Una variación del método de implementación gradual, llamado lanzamiento de valores controlados, implica la implementación de la nueva versión de software en un porcentaje muy pequeño de servidores al principio. De esta manera, puede observar cómo se comporta el software en producción en unos pocos servidores, mientras se minimiza el impacto de los cambios importantes. Si hay una tasa elevada de errores de una implementación de valores controlados, el software se revierte. De lo contrario, el porcentaje de servidores con la nueva versión aumentará gradualmente.

AWS Elastic Beanstalk ha seguido el patrón de implementación gradual con dos opciones de implementación, continua y continua con lote adicional. Estas opciones permiten que la aplicación se amplíe primero antes de dejar los servidores fuera de servicio, lo que preserva toda la capacidad durante la implementación. AWS CodeDeploy logra este patrón como una variación de una implementación in situ con patrones como OneAtaTime y HalfAtime.

Implementación inmutable y azul/verde

El patrón inmutable especifica una implementación de código de aplicación iniciando un conjunto de servidores completamente nuevo con una nueva configuración o versión del código de aplicación.

Este patrón aprovecha la capacidad de la nube para crear los nuevos recursos del servidor con llamadas simples a la API.

La estrategia de implementación azul/verde es un tipo de implementación inmutable que también requiere la creación de otro entorno. Una vez que el nuevo entorno esté activo y haya superado todas las pruebas, el tráfico se traslada a esta nueva implementación. Fundamentalmente, el entorno antiguo, es decir, el entorno “azul”, se mantiene inactivo en caso de que se necesite una reversión.

AWS Elastic Beanstalk admite patrones de implementación inmutable y azul/verde. AWS CodeDeploy también es compatible con el patrón azul/verde. Para obtener más información sobre cómo los servicios de AWS logran estos patrones inmutables, consulte el documento técnico Implementaciones azul/verde en AWS.

Cambios de esquema de base de datos

Es común que el software moderno tenga una capa de base de datos. Por lo general, se utiliza una base de datos relacional, que almacena tanto los datos como la estructura de los datos. A menudo es necesario modificar la base de datos en el proceso de entrega continua. El manejo de los cambios en una base de datos relacional requiere una consideración especial y ofrece otros desafíos además de los que se presentan al implementar binarios de aplicaciones. Por lo general, cuando se actualiza un binario de aplicación, se detiene la aplicación, se actualiza y, a continuación, se inicia de nuevo. En realidad, no hay que preocuparse por el estado de la aplicación, que se maneja fuera de la aplicación.

Al actualizar bases de datos, hay que tener en cuenta el estado porque una base de datos contiene mucho estado pero relativamente poca lógica y estructura.

El esquema de base de datos antes y después de aplicar un cambio debe considerarse como versiones diferentes de la base de datos. Puede utilizar herramientas como Liquibase y Flyway para administrar las versiones.

En general, esas herramientas emplean alguna variante de los siguientes métodos:

- Agregar una tabla a la base de datos en la que hay almacenada una versión de base de datos.
- Realizar un seguimiento de los comandos de cambio de base de datos y agruparlos en conjuntos de cambios con versiones. En el caso de Liquibase, estos cambios se almacenan en archivos XML. Flyway emplea un método ligeramente diferente en el que los conjuntos de cambios se manejan como archivos SQL separados u ocasionalmente como clases Java separadas para transiciones más complejas.
- Cuando se le pide a Liquibase que actualice una base de datos, mira la tabla de metadatos y determina qué conjuntos de cambios se deben ejecutar para actualizar la base de datos con la última versión.

Resumen de las prácticas recomendadas

Las siguientes son algunas de las mejores prácticas recomendadas que se deben y no se deben hacer para la CI/CD.

Hacer:

- Tratar la infraestructura como código
  - Usar el control de versiones para el código de la infraestructura.
  - Hacer uso de sistemas de seguimiento de errores y tickets.
  - Hacer que los compañeros revisen los cambios antes de aplicarlos
  - Establecer diseños/patrones de código de infraestructura.
  - probar cambios en la infraestructura, como cambios de código.
- Poner a los desarrolladores en equipos integrados de no más de 12 miembros autosuficientes.
- Hacer que todos los desarrolladores envíen código al tronco principal con frecuencia, sin ramificaciones de características de larga duración.
- Adoptar de manera consistente un sistema de compilación como Maven o Gradle en toda la organización y estandarizar las compilaciones.
- Hacer que los desarrolladores creen pruebas unitarias para cubrir el 100 % de la base de código.
- Asegurarse de que las pruebas unitarias representen el 70 % del total de las pruebas en duración, número y alcance.
- Asegurarse de que las pruebas unitarias estén actualizadas y no se descuiden. Los errores de pruebas unitarias deben repararse, no pasarse por alto.
- Tratar la configuración de entrega continua como código.
- Establecer controles de seguridad basados en roles (es decir, quién puede hacer qué y cuándo).
  - Monitorear/hacer un seguimiento de todos los recursos posibles.
  - Alertar sobre los servicios, la disponibilidad y los tiempos de respuesta.
  - Capturar, aprender y mejorar.
  - Compartir el acceso con todos los miembros del equipo.
  - Planificar métricas y monitorear en el ciclo de vida.
- Mantener y hacer un seguimiento de las métricas estándar.
  - Número de compilaciones.

  - Número de implementaciones.
  - Tiempo medio para que los cambios lleguen a la producción.
  - Tiempo medio desde la primera etapa de la canalización en cada etapa.
  - Número de cambios que llegan a la producción.
  - Tiempo medio de compilación.
- Usar varias canalizaciones distintas para cada ramificación y equipo.

No hacer:

- Tener sucursales de larga duración con fusiones grandes y complicadas.
- Realizar pruebas manuales.
- Tener procesos de aprobación, puertas, revisiones de códigos y revisiones de seguridad manuales.

Conclusión

La integración continua y la entrega continua proporcionan un escenario ideal para los equipos de aplicaciones de su organización. Sus desarrolladores simplemente insertan el código en un repositorio. Este código se integrará, se probará, se implementará, se probará nuevamente, se fusionará con la infraestructura, pasará por revisiones de seguridad y calidad y estará listo para implementarse con una confianza extremadamente alta.

Cuando se utiliza CI/CD, se mejora la calidad del código y las actualizaciones de software se entregan rápidamente y con una alta confianza de que no habrá cambios importantes. El impacto de cualquier lanzamiento se puede correlacionar con los datos de producción y operaciones. También se puede utilizar para planificar el siguiente ciclo, una práctica de DevOps vital en la transformación de la nube de su organización.

Documentación adicional

Para obtener más información sobre los temas tratados en este documento técnico, consulte los siguientes documentos técnicos de AWS:

- Información general sobre las opciones de implementación en AWS
- Implementaciones azul/verde en AWS
- Configuración de la canalización de CI/CD mediante la integración de Jenkins con AWS CodeBuild y AWS CodeDeploy
- Microservicios en AWS
- Docker on AWS: Running Containers in the Cloud

Colaboradores

Las siguientes personas y organizaciones contribuyeron a redactar este documento:

- Amrish Thakkar, arquitecto principal de soluciones, AWS
- David Stacy, consultor sénior de DevOps, Servicios profesionales de AWS
- Asif Khan, arquitecto de soluciones, AWS
- Xiang Shen, arquitecto sénior de soluciones, AWS

Revisiones del documento

Para recibir notificaciones sobre las actualizaciones de este documento técnico, suscríbase a la fuente RSS.

update-history-change	update-history-description	update-history-date
Publicación inicial	Documento técnico publicado por primera vez	27 de octubre de 2021
Publicación inicial	Documento técnico publicado por primera vez	1 de junio de 2017

Avisos

Los clientes son responsables de realizar sus propias evaluaciones de la información contenida en este documento. Este documento: (a) solo tiene fines informativos, (b) representa las prácticas y las ofertas de productos vigentes de AWS, que están sujetas a cambios sin previo aviso, y (c) no crea ningún compromiso ni garantía de AWS y sus empresas afiliadas, proveedores o concesionarios de licencias. Los productos o servicios de AWS se proporcionan “tal cual”, sin garantías, representaciones ni condiciones de ningún tipo, ya sean explícitas o implícitas. Las responsabilidades y obligaciones de AWS en relación con sus clientes se rigen por los acuerdos de AWS, y este documento no modifica ni forma parte de ningún acuerdo entre AWS y sus clientes.