Automatización de implementaciones seguras y sin intervención

Clare Liguori

En mi entrevista de trabajo para Amazon, me aseguré de preguntar a uno de los entrevistadores: "¿Cuán a menudo realizan implementaciones en la producción?". En ese momento, trabajaba en un producto para el cual se realizaba un lanzamiento principal una o dos veces al año, pero, algunas veces, tenía lanzar una pequeña corrección entre los lanzamientos grandes. Por cada corrección que generaba, pasaba horas preparando con cuidado su lanzamiento. Luego, revisaba los registros y las métricas frenéticamente para verificar que no hubiese nada roto después de la implementación que me obligara a revertirla.

Había leído que Amazon efectuaba implementaciones continuas, por lo que, cuando me entrevistaron, quería saber cuánto tiempo pasaría completando tareas de administración y observación de las implementaciones como desarrolladora en Amazon. El entrevistador me explicó que las canalizaciones de implementación continua insertaban los cambios de manera automática en la producción varias veces por día. Cuando le pregunté cuánto tiempo de su día tenía que dedicar a revisar con cuidado cada una de esas implementaciones y observar los registros y las métricas para detectar problemas tal como yo lo hacía, me contestó que, por lo general, nada de tiempo. Comentó que como las canalizaciones hacían ese trabajo por su equipo, la mayoría de las implementaciones no necesitaba que nadie las vigilara de forma activa. "¡Guau!", respondí. Una vez que formaba parte del equipo de Amazon, me emocionaba descubrir cómo funcionaban exactamente esas implementaciones automatizadas "sin intervención".

Manera en que las implementaciones seguras y continuas permiten optimizar el tiempo del desarrollador

Desde ese entonces, he experimentado de primera mano la manera en que Amazon configura canalizaciones de implementación continua para ayudarnos a realizar esta tarea de forma rápida y segura. Llegué a apreciar cómo nuestras prácticas de seguridad en la implementación continua liberan al desarrollador del trabajo de las implementaciones. Cuando introduzco el código de producción en la ramificación principal del repositorio de código fuente de mi servicio, por lo general, me olvido de ello y sigo con mi próxima tarea, mientras la canalización de mi equipo se encarga de llevar ese cambio a la producción. La canalización automatiza por completo el lanzamiento de mi cambio en el código en un servicio de producción, lo que significa que la última vez que yo o cualquier otro desarrollador haya tocado o revisado parte del código será el momento en que se incluya en el repositorio de código fuente.

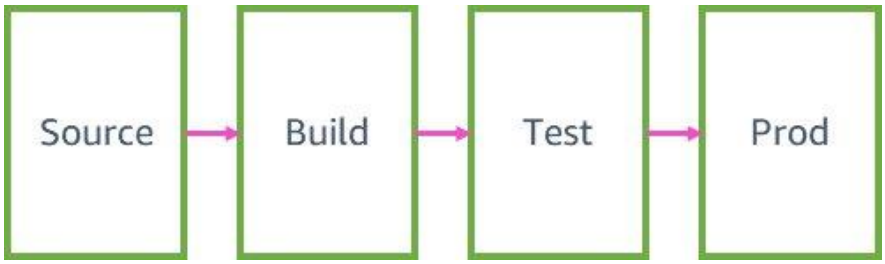
Mi equipo configura la canalización con pasos automatizados que implementan los cambios de manera segura en la producción, de modo que nosotros no tengamos que controlar cada implementación. La canalización somete los últimos cambios a un conjunto de pruebas y comprobaciones de seguridad de las implementaciones. Estos pasos automatizados evitan que los defectos que afectan a los clientes lleguen a la producción y limitan el impacto que puedan ejercer sobre los clientes, si es que llegan a la producción. Como desarrolladora, puedo confiar en que la canalización implementará de manera cautelosa y segura el cambio en la producción por mí, sin necesidad de que revise la implementación de forma activa.

La trayectoria hacia la entrega continua

Amazon no comenzó a operar practicando la entrega continua. Los desarrolladores solían pasar horas y días administrando las implementaciones del código en la producción. Adoptamos la entrega continua en toda la empresa como una forma de automatizar y estandarizar el modo en el que implementábamos el software y la forma de reducir la cantidad de tiempo que se necesitaba para llevar los cambios a la producción. La cantidad de mejoras en nuestro proceso de lanzamiento aumentó de manera progresiva con el paso del tiempo. Identificamos los riesgos de la implementación y encontramos formas de mitigarlos a través de una nueva automatización de la seguridad en las canalizaciones. Continuamos trabajando en el proceso de lanzamiento mediante la identificación de nuevos riesgos y nuevas formas de mejorar la seguridad de las implementaciones. Para obtener más información acerca de nuestra trayectoria hacia la entrega continua y cómo seguimos mejorando, consulte el artículo de Builders' Library Evolución más rápida con la entrega continua.

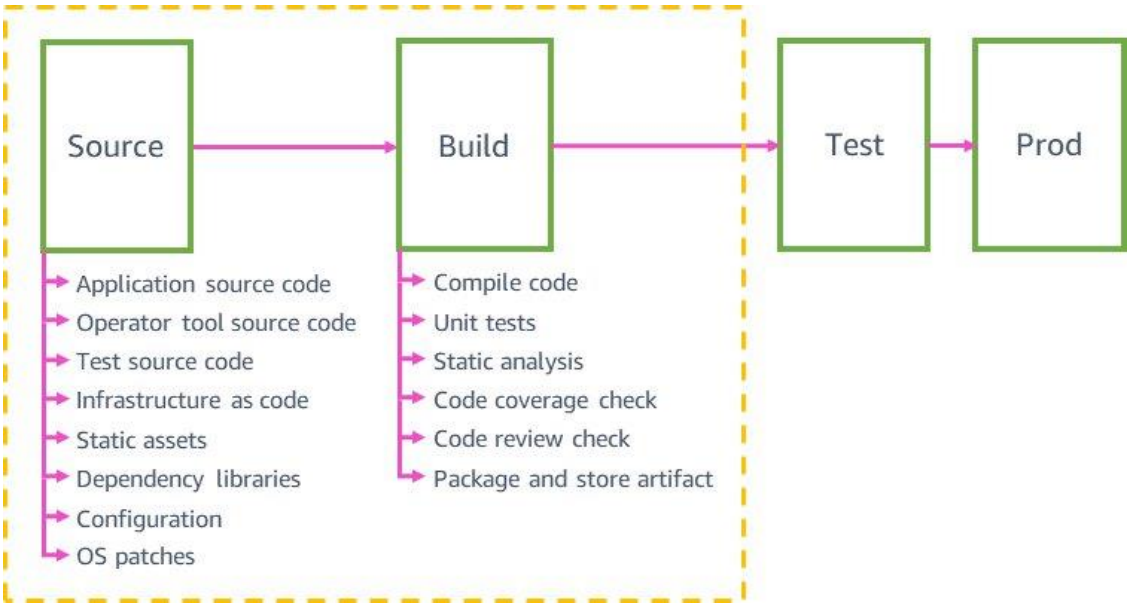
Las cuatro fases de la canalización

En este artículo, recorremos los pasos que atraviesa un cambio de código en una canalización de Amazon en su camino hacia la producción. Una canalización de entrega continua típica consta de cuatro fases principales: fuente, compilación, prueba y producción (prod.). Abordaremos los detalles de lo que sucede en cada una de estas fases de las canalizaciones en relación con servicios típicos de AWS y le proporcionaremos un ejemplo de cómo el equipo de un servicio típico de AWS puede configurar una de estas canalizaciones.



Fuente y compilación

El siguiente diagrama le brinda información general sobre los pasos de fuente y compilación que tal vez encuentre en las canalizaciones de los equipos de servicios típicos de AWS.



Fuentes de las canalizaciones

En Amazon, las canalizaciones validan de forma automática e implementan de manera segura cualquier tipo de cambio de fuente en la producción, no solo los cambios en el código de aplicación. Pueden validar e implementar cambios en la fuente, como los recursos estáticos, las herramientas, las pruebas, la infraestructura y la configuración de sitios web, y el sistema operativo (SO) subyacente de la aplicación. Todos estos cambios se someten al control de versiones en los repositorios de código fuente individuales. Las dependencias de código fuente, como las bibliotecas, los lenguajes de programación y los parámetros, tales como los ID de AMI, se actualizan automáticamente a la versión más reciente por lo menos una vez por semana.

Estas fuentes se implementan en canalizaciones individuales con los mismos mecanismos de seguridad (como las restauraciones automáticas) que utilizamos para implementar código de aplicación. Por ejemplo, los valores de configuración de servicios que pueden cambiar durante el tiempo de ejecución (como los aumentos del límite de velocidad de la API y los interruptores de características) se implementan de forma automática en una canalización de configuración dedicada. Los cambios de fuente se revierten de manera automática si causan algún problema en la producción del servicio (como errores en el análisis de un archivo de configuración).

Un microservicio típico puede tener una canalización de código de aplicación, una canalización de infraestructura, una canalización de implementación de parches en el sistema operativo, una canalización de interruptor de características o configuraciones, y una canalización de herramientas del operador. Contar con varias canalizaciones para el mismo microservicio nos ayuda a implementar los cambios en la producción más rápido. Los cambios en los códigos de aplicación que no obtienen resultados correctos en las pruebas de integración y bloquean la canalización de aplicación no afectan a las demás canalizaciones. Por ejemplo, no bloquean el paso de los cambios en los códigos de infraestructura a la etapa de producción en el marco de la canalización de infraestructura. Todas las canalizaciones del mismo microservicio suelen tener aspecto similar. Por ejemplo, la canalización de

interruptor de características utiliza las mismas técnicas de implementación segura que la canalización de código de aplicación. Esto se debe a que un cambio con errores en la configuración de interruptor de características puede tener efectos negativos en la producción, de la misma manera que lo haría un cambio con errores en el código de aplicación.

Revisión de código

Todos los cambios que pasan a la producción comienzan con una revisión de código y deben ser aprobados por un miembro del equipo antes de incorporarlos a la ramificación *principal* (nuestra versión de "maestra" o "troncal"), la cual inicia de forma automática la canalización. La canalización hace cumplir el requisito de que un miembro del equipo del servicio de esa canalización debe revisar y aprobar el código de todas las confirmaciones que se encuentren en la ramificación principal. La canalización impedirá la implementación de las confirmaciones que no se hayan revisado.

En el caso de las canalizaciones completamente automatizadas, la revisión de código es la última revisión y aprobación manuales a las que se somete un cambio de código por parte de un ingeniero antes de implementarlo en la producción, por lo que se trata de un paso clave. Los revisores de código evalúan que el código sea correcto y si el cambio se puede implementar de manera segura en la producción. Evalúan si el código tiene pruebas suficientes (pruebas de unidades, de integración y de valor controlado), si cuenta con los instrumentos necesarios para el monitoreo de la implementación y si se puede restaurar de forma segura. Algunos equipos utilizan una lista de verificación personalizada, como la del siguiente ejemplo, que se agrega automáticamente a cada revisión de código del equipo para verificar de forma explícita los asuntos relacionados con la seguridad de las implementaciones.

Ejemplo de lista de verificación para la revisión de código

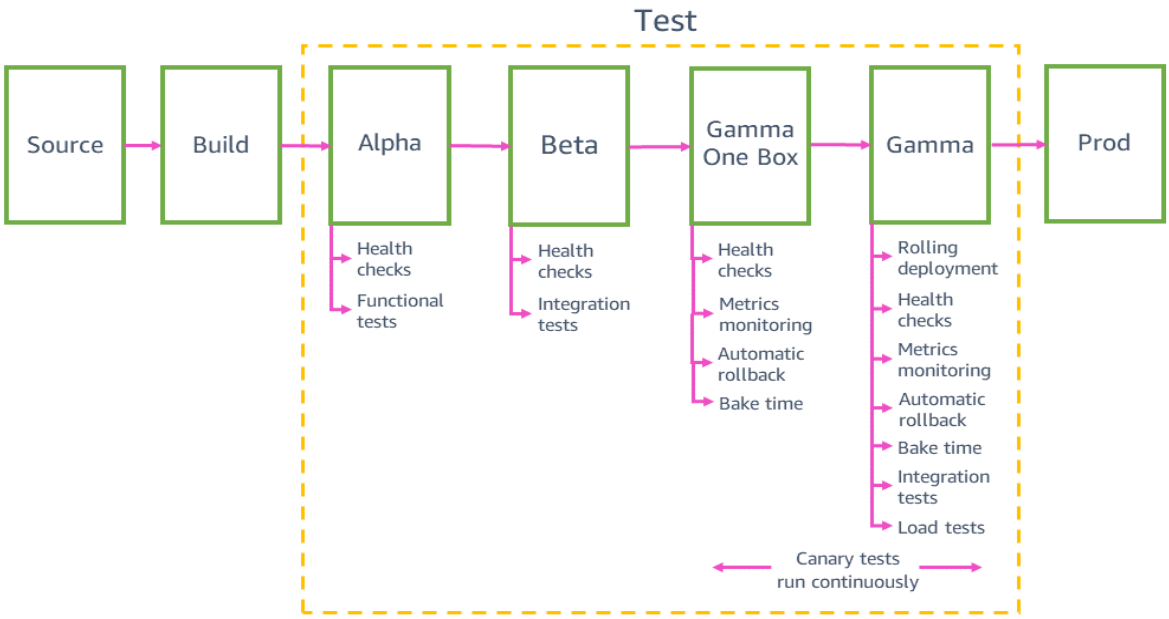
```
## Testing
[ ] Did you write new unit tests for this change?
[ ] Did you write new integration tests for this change?

Include the test commands you ran locally to test this change:
```
mvn test && mvn verify
```

## Monitoring
[ ] Will this change be covered by our existing monitoring?
 (no new canaries/metrics/dashboards/alarms are required)
[ ] Will this change have no (or positive) effect on resources and/or
limits?
 (including CPU, memory, AWS resources, calls to other services)
[ ] Can this change be deployed to Prod without triggering any alarms?

## Rollout
[ ] Can this change be merged immediately into the pipeline upon approval?
[ ] Are all dependent changes already deployed to Prod?
[ ] Can this change be rolled back without any issues after deployment to
Prod?
```

Compilación y pruebas de unidades

En la etapa de compilación, se compila el código y se lo somete a la prueba de unidades. Las herramientas y la lógica de compilación pueden variar de un lenguaje a otro e, incluso, de un equipo a otro. Por ejemplo, los equipos pueden elegir los marcos de prueba de unidades, las herramientas de análisis de código fuente (linters) y las herramientas de análisis estático que mejor funcionen para ellos. Además, pueden elegir la configuración de esas herramientas, como la cobertura de código mínima aceptable en el marco de prueba de unidades. Las herramientas y los tipos de pruebas que se ejecutan también varían en función del tipo de código que implementa la canalización. Por ejemplo, las pruebas de unidades se utilizan para el código de aplicación, y las herramientas lint se utilizan para las plantillas de infraestructura como código. Todas las compilaciones se ejecutan sin acceso a la red con el fin de aislarlas y fomentar la capacidad de reproducción. Por lo general, las pruebas de unidades simulan todas las llamadas de la API a las dependencias, como otros servicios de AWS. Las interacciones con las dependencias "en directo" no simuladas se prueban más tarde en la canalización, en las pruebas de integración. En comparación con las pruebas de integración, las pruebas de unidades con dependencias simuladas pueden ejercer casos de borde, como errores inesperados que surgen de las llamadas de la API, y garantizan una gestión adecuada de los errores en el código. Cuando se completa la compilación, el código compilado se empaqueta y firma.

Implementaciones de prueba en entornos de preproducción

Antes de efectuar implementaciones en la producción, la canalización implementa y valida los cambios en varios entornos de preproducción, por ejemplo, alfa, beta y gamma. Alfa y beta validan que el código más nuevo funcione como se espera mediante la ejecución de pruebas funcionales de la API y pruebas de integración completas. Gamma valida que el código sea funcional y se pueda implementar de manera segura en la producción. Gamma es lo más parecido posible al entorno de producción, dado que incluye la misma configuración de implementación, el mismo monitoreo, las mismas alarmas y las mismas pruebas continuas de valor controlado que el entorno de producción. Gamma también se implementa en varias regiones de AWS para poder detectar cualquier posible efecto derivado de las diferencias regionales.



Pruebas de integración

Las pruebas de integración nos ayudan a usar un servicio de manera automática, tal cual lo hacen los clientes, como parte de la canalización. Estas pruebas ponen en uso la pila completa llamando a las API reales que se ejecutan en infraestructura real, en cada etapa de preproducción y por todas las situaciones significativas de los clientes. El objetivo de la prueba de integración es captar cualquier comportamiento inesperado o incorrecto del servicio antes de efectuar la implementación en la producción.

Mientras las pruebas de unidades se ejecutan en dependencias simuladas, las pruebas de integración se ejecutan en un sistema de preproducción que llama a dependencias reales, lo cual valida las suposiciones de las simulaciones respecto de cómo se comportan esas dependencias. Las pruebas de integración validan el comportamiento de las API individuales respecto de entradas diferentes. Además, validan flujos de trabajo completos que unen varias API, como la creación de un nuevo recurso, la descripción del nuevo recurso hasta que esté listo y, luego, el uso del recurso.

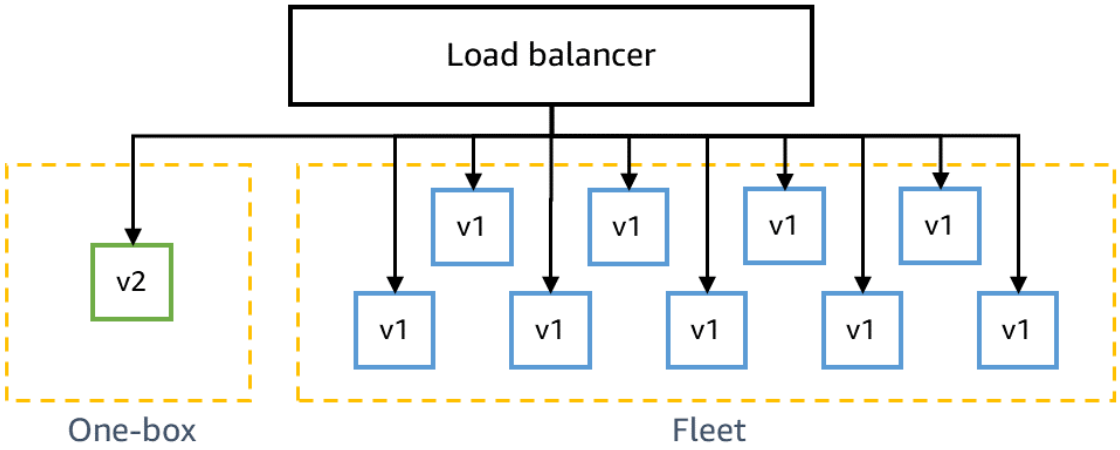
Las pruebas de integración ejecutan tanto casos de prueba positivos como negativos, por ejemplo, la provisión de entradas inválidas a una API y la verificación de que se obtenga un error de “entrada inválida” como se espera. Algunas canalizaciones ejecutan una prueba de fuzzing, con el fin de generar muchas entradas posibles a la API y validar que ellas no causen errores internos en el servicio. Algunas canalizaciones también ejecutan una prueba corta de carga en la etapa de preproducción para asegurarse de que los últimos cambios no generen latencia ni regresión en el rendimiento con niveles reales de carga.

Compatibilidad retroactiva y prueba de caja individual

Antes de efectuar la implementación en la producción, tenemos que asegurarnos de que el último código sea compatible con versiones anteriores y se pueda implementar de manera segura junto con

el código actual. Por ejemplo, debemos detectar si el último código escribe los datos en un formato que el código actual no puede procesar. La etapa de *caja individual* en gamma implementa el último código en la unidad de implementación más pequeña, como una sola máquina virtual o un único contenedor, o en un pequeño porcentaje de invocaciones de funciones de AWS Lambda. Esta implementación en caja individual deja que el resto del entorno gamma continúe con el código actual durante un periodo determinado, como 30 minutos o una hora. No es necesario que el tráfico se dirija específicamente a esa caja. Se puede agregar al mismo balanceador de carga o analizar la misma cola que el resto del entorno gamma. Por ejemplo, en un entorno gamma de diez contenedores detrás de un balanceador de carga, la caja individual recibe el 10 % del tráfico de gamma generado por las pruebas continuas de valor controlado. La implementación de caja individual monitorea los índices de éxito de las pruebas de valor controlado y las métricas del servicio para detectar cualquier efecto producido por la implementación o por tener una flota "mixta" implementada al lado.

El siguiente diagrama muestra el estado de un entorno gama después de haber implementado código nuevo en la etapa de caja individual, pero sin haberlo implementado todavía el resto de la flota gamma:



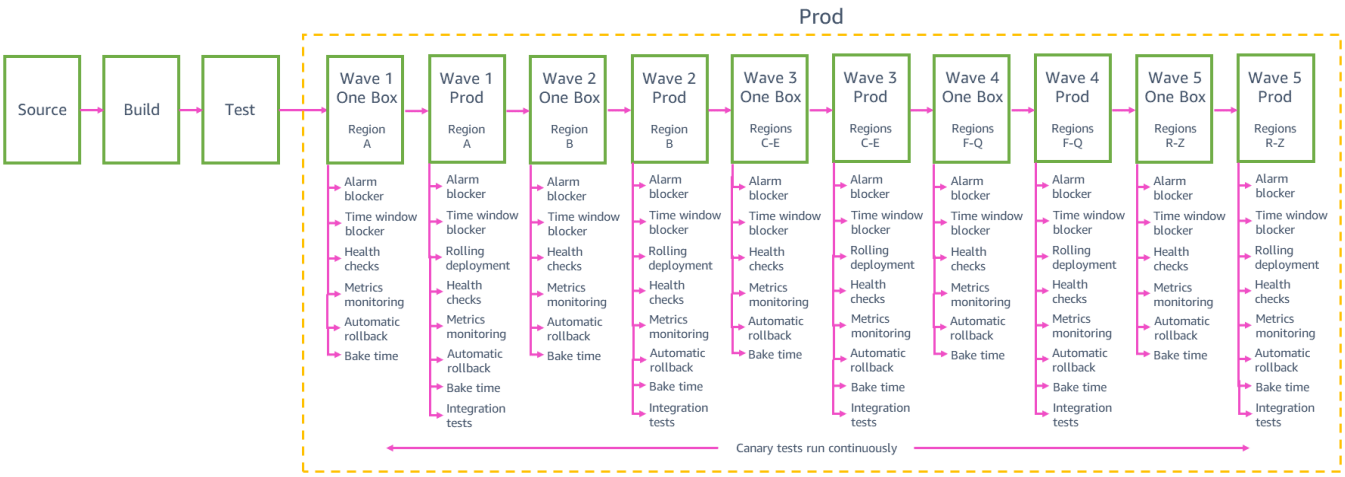
También debemos asegurarnos de que el último código sea compatible con las dependencias anteriores, por ejemplo, si se debe efectuar un cambio en los microservicios de acuerdo con un orden específico. Por lo general, los microservicios en los entornos de preproducción llaman al punto de enlace de producción de cualquier servicio que sea de propiedad de otro equipo, como Amazon Simple Storage Service (S3) o Amazon DynamoDB, pero llaman al punto de enlace de preproducción de los demás microservicios del equipo del servicio en la misma etapa. Por ejemplo, el microservicio A del equipo en gamma llama al microservicio B del mismo equipo en gamma, pero llama al punto de enlace de producción de Amazon S3.

Algunas canalizaciones también ejecutan pruebas de integración otra vez en una etapa de compatibilidad retroactiva independiente que llamamos *zeta*. Esta representa un entorno independiente en el cual cada microservicio llama solo a puntos de enlace de producción. De esta manera, verifican en varios microservicios que los cambios que se insertarán en la producción sean compatibles con el código actualmente implementado en el entorno de producción. Por ejemplo, el microservicio A en zeta llama al punto de enlace de producción del microservicio B y al punto de enlace de producción de Amazon S3.

Si desea obtener una descripción de las estrategias para escribir e implementar cambios que sean compatibles con versiones anteriores, consulte el artículo de Builders' Library [Garantía de la seguridad en las restauraciones durante las implementaciones](#).

Implementaciones en la producción

Nuestro objetivo n.° 1 para las implementaciones en la producción en AWS es evitar el impacto negativo sobre varias regiones al mismo tiempo y múltiples zonas de disponibilidad de la misma región. Limitar el alcance de cada implementación individual restringe el posible impacto sobre los clientes de las implementaciones fallidas en la producción y evita que se produzcan efectos en varias zonas de disponibilidad o varias regiones. A fin de limitar el alcance de las implementaciones automáticas, dividimos la fase de producción de la canalización en muchas etapas e implementaciones en regiones individuales. Los equipos dividen las implementaciones regionales en otras con un alcance aún más reducido mediante implementaciones en zonas de disponibilidad individuales o en particiones individuales internas (llamadas *celdas*) del servicio en la canalización, para limitar todavía más el alcance de los posibles efectos de una implementación en la producción con errores.



Implementaciones escalonadas

Cada equipo necesita equilibrar la seguridad de las implementaciones de alcance reducido con la velocidad a la cual podemos entregar los cambios a los clientes en todas las regiones. La implementación de cambios de a uno por vez en 24 regiones o 76 zonas de disponibilidad a través de la canalización conlleva el menor riesgo de impacto generalizado, pero es posible que la canalización tarde semanas en entregar un cambio a los clientes a nivel mundial. Descubrimos que agrupar las implementaciones en “oleadas” de tamaño creciente, como se puede ver en el ejemplo anterior de canalización de producción, nos ayuda a lograr un buen equilibrio entre el riesgo y la velocidad de las implementaciones. Cada etapa de las oleadas de la canalización organiza las implementaciones en un grupo de regiones, con cambios que pasan de una oleada a otra. Los cambios nuevos pueden pasar a la fase de producción de la canalización en cualquier momento. Luego de que un conjunto de cambios pasa del primer paso al segundo en la oleada n.° 1, el siguiente conjunto de cambios de gamma pasa

al primer paso de la oleada n.° 1, de manera que no terminamos con grandes paquetes de cambios a la espera de implementación en la producción.

Las primeras dos oleadas de la canalización generan la mayor parte de la confianza en el cambio. La primera oleada se implementa en una región con una cantidad reducida de solicitudes para limitar el posible impacto de la primera implementación en la producción del nuevo cambio. La oleada se implementa solo en una zona de disponibilidad (o celda) por vez dentro de esa región para insertar con cautela el cambio en toda la región. La segunda oleada se implementa en una zona de disponibilidad (o celda) a la vez de una región con un número alto de solicitudes, donde es muy probable que los clientes ejecuten todas las rutas del código nuevo y donde obtengamos una buena validación de los cambios.

Luego de tener mayor confianza en la seguridad del cambio desde las implementaciones de oleadas de canalizaciones iniciales, podemos implementar en cada vez más regiones simultáneas de la misma oleada. Por ejemplo, la canalización de producción de muestra anterior se implementa en tres regiones de la oleada 3; luego, en un máximo de 12 regiones de la oleada 4; y por último, en las regiones restantes de la oleada 5. La cantidad exacta de regiones y la elección de ellas en cada una de estas oleadas y la cantidad de oleadas en la canalización de un equipo de servicio dependen de los patrones y la escala del uso particular del servicio. Las oleadas posteriores en la canalización también nos ayudan a lograr nuestro objetivo de evitar un impacto negativo en varias zonas de disponibilidad en la misma región. Cuando una oleada se implementa en varias regiones simultáneas, sigue el mismo comportamiento cauteloso de despliegue para cada región que se utilizó en las oleadas iniciales. Cada paso en la oleada solamente se implementa en una sola celda o zona de disponibilidad de cada región en la oleada.

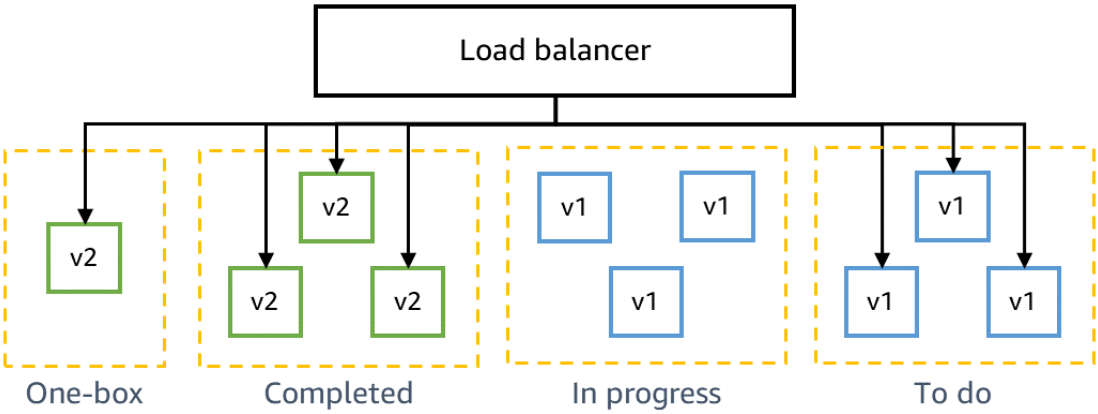
Implementaciones de caja individual y continuas

Las implementaciones en cada oleada de producción comienzan con una etapa de caja individual. Como en la etapa de caja individual de gamma, cada etapa de caja individual de producción implementa el código más reciente en una *caja* individual (una sola máquina virtual, un único contenedor o un pequeño porcentaje de invocaciones de funciones de Lambda) en cada una de las regiones o las zonas de disponibilidad de la oleada. La implementación de caja individual de producción minimiza el impacto posible de los cambios en la oleada mediante la limitación inicial de las solicitudes que atiende el código nuevo en esa oleada. Normalmente, la caja individual atiende como máximo un diez por ciento del total de solicitudes de la región o la zona de disponibilidad. Si el cambio produce un impacto negativo en la caja individual, la canalización automáticamente restaura el contenido anterior y no extiende ese cambio al resto de las etapas de producción.

Después de la etapa de caja individual, la mayoría de los equipos utiliza implementaciones continuas que se llevan a cabo en la flota principal de producción de la oleada. Las implementaciones continuas garantizan que el servicio cuente con suficiente capacidad para atender la carga de producción en toda la implementación. Controlan la velocidad a la que se *pone en funcionamiento* el nuevo código (es decir, cuando comienza a atender el tráfico de producción) para limitar el impacto de los cambios. En una implementación continua típica en una región, a lo sumo el 33 % de las cajas de servicio en esa región (contenedores, invocaciones de Lambda o software que se ejecuta en máquinas virtuales) se reemplaza con el código nuevo.

Durante una implementación, el sistema de implementación elige en primer lugar un lote inicial de hasta un 33 por ciento de cajas para reemplazar con el código nuevo. Durante el reemplazo, al menos el 66 por ciento de la capacidad total se encuentra en buen estado y atiende solicitudes. Todos los servicios se escalan para soportar la pérdida de una zona de disponibilidad en la región, por lo que sabemos que el servicio aún puede atender la carga de producción a esta capacidad. Una vez que el sistema de implementación determina que una caja en el lote inicial de cajas no presenta errores en las comprobaciones de estado, se puede reemplazar una caja de la flota restante con el nuevo código, y así sucesivamente. Mientras tanto, se mantiene un mínimo de 66 % de capacidad para atender solicitudes en todo momento. Para limitar aún más el impacto de los cambios, las canalizaciones de algunos equipos implementan hasta solo un 5 % de sus cajas a la vez. Sin embargo, luego llevan a cabo *restauraciones rápidas*, en las que el sistema reemplaza el 33 % de las cajas a la vez con el código anterior para acelerar la restauración.

El siguiente diagrama muestra el estado de un entorno de producción en medio de una implementación continua. El código nuevo se ha implementado en la etapa de caja individual y en el primer lote de la flota principal de producción. Otro lote se ha eliminado del balanceador de carga y se desactiva para su posterior reemplazo.



Monitoreo y restauración automática de las métricas

Las implementaciones automatizadas en las canalizaciones no suelen tener un desarrollador que se dedique a observar cada implementación en la producción, comprobar las métricas y llevar a cabo una restauración manual si encuentra problemas. Estas implementaciones se realizan sin ningún tipo de intervención manual. El sistema de implementación monitorea una alarma de forma activa para determinar si necesita restaurar automáticamente una implementación. Una restauración revertirá el entorno a la imagen del contenedor, al paquete de implementación de las funciones de AWS Lambda o al paquete interno de implementación que se utilizó previamente. Nuestros paquetes internos de implementación son similares a las imágenes de contenedor, ya que los paquetes son inmutables y utilizan una suma de comprobación para verificar su integridad.

Cada microservicio de una región suele tener una alarma de nivel alto de gravedad que se activa en los límites para las métricas que tienen un impacto en los clientes del servicio (como las tasas de errores y la latencia alta) y en las métricas en buen estado del sistema (como la utilización de CPU), tal y como

se muestra en el ejemplo siguiente. Esta alarma de nivel alto de gravedad se utiliza para llamar al ingeniero de turno y para restaurar automáticamente el servicio si hay una implementación en progreso. A menudo, la restauración ya se encuentra en progreso cuando el ingeniero de turno recibe la llamada y se ocupa del asunto.

Ejemplo de alarma de microservicio con nivel alto de gravedad

```
ALARM("FrontEndApiService_High_Fault_Rate") OR
ALARM("FrontEndApiService_High_P50_Latency") OR
ALARM("FrontEndApiService_High_P90_Latency") OR
ALARM("FrontEndApiService_High_P99_Latency") OR
ALARM("FrontEndApiService_High_Cpu_Usage") OR
ALARM("FrontEndApiService_High_Memory_Usage") OR
ALARM("FrontEndApiService_High_Disk_Usage") OR
ALARM("FrontEndApiService_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_High_Failing_Health_Checks")
```

Los cambios que introduce una implementación pueden afectar los microservicios anteriores y posteriores, por lo que el sistema de implementación debe monitorear la alarma de nivel alto de gravedad para el microservicio en proceso de implementación *y* monitorear las alarmas de nivel alto de gravedad para los otros microservicios del equipo, con el fin de determinar en qué momento llevar a cabo la restauración. Los cambios implementados también pueden afectar las métricas de las pruebas continuas de valor controlado, por lo que el sistema de implementación también debe realizar monitoreos para detectar pruebas de valor controlado que tengan errores. Para llevar a cabo una restauración de forma automática en todas estas posibles áreas de impacto, los equipos crean alarmas agrupadas de nivel alto de gravedad para que las monitoree el sistema de implementación. Las alarmas agrupadas de nivel alto de gravedad contemplan el estado de todas las alarmas de nivel alto de gravedad del microservicio individual del equipo y el estado de las alarmas de valor controlado en un solo estado agrupado, como en el ejemplo siguiente. Si alguna de las alarmas de nivel alto de gravedad para los microservicios del equipo se activa, todas las implementaciones en curso del equipo en todos los microservicios de esa región se restaurarán de forma automática.

Ejemplo de alarma agrupada de nivel alto de gravedad para la restauración

```
ALARM("FrontEndApiService_High_Severity") OR
ALARM("BackendApiService_High_Severity") OR
ALARM("BackendWorkflows_High_Severity") OR
ALARM("Canaries_High_Severity")
```

La etapa de caja individual atiende un pequeño porcentaje del tráfico total, por lo que existe la posibilidad de que los problemas que se presenten por una implementación de caja individual no activen la alarma de nivel alto de gravedad para la restauración del servicio. Para captar y restaurar los cambios que causan problemas en la etapa de caja individual antes de que lleguen al resto de las etapas de producción, las etapas de caja individual también restauran las métricas que solo afectan la caja individual. Por ejemplo, llevan a cabo la restauración en la tasa de errores de las solicitudes que la caja individual atendió específicamente, lo que comprende un pequeño porcentaje del número total de solicitudes.

Ejemplo de alarma de caja individual para la restauración

```
ALARM("High_Severity_Aggregate_Rollback_Alarm") OR
ALARM("FrontEndApiService_OneBox_High_Fault_Rate") OR
ALARM("FrontEndApiService_OneBox_High_P50_Latency") OR
```

```
ALARM("FrontEndApiService_OneBox_High_P90_Latency") OR
ALARM("FrontEndApiService_OneBox_High_P99_Latency") OR
ALARM("FrontEndApiService_OneBox_High_Cpu_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Memory_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Disk_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_OneBox_Failing_Health_Checks")
```

Además de restaurar según las alarmas definidas por el equipo de servicio, nuestro sistema de implementación también puede detectar y restaurar de forma automática ante anomalías en métricas comunes emitidas por nuestro marco de servicio web interno. La mayoría de nuestros microservicios emiten métricas como recuento, latencia de solicitudes y recuento de errores en un formato estándar. Mediante la utilización de estas métricas estándar, el sistema de implementación puede llevar a cabo restauraciones automáticas si hay anomalías en las métricas durante una implementación. Por ejemplo: cuando el recuento de solicitudes repentinamente baja a cero, o cuando la latencia o el número de errores se incrementa más de lo normal.

Tiempo de procesamiento

En algunas ocasiones, un impacto negativo causado por una implementación no se pone de manifiesto inmediatamente. Se *procesa lentamente*. Esto significa que no se muestra de inmediato durante la implementación, en especial si, en ese momento, el servicio tiene una carga liviana. Promover el cambio a la próxima etapa de canalización inmediatamente después de que la implementación se complete puede llegar a afectar varias regiones antes de que dicho impacto se haga evidente en la primera región. Antes de promover un cambio a la próxima etapa de producción, cada etapa de producción en la canalización debe dejar pasar un *tiempo de procesamiento*, durante el cual la canalización continúa monitoreando la alarma agrupada de nivel alto de gravedad del equipo a fin de detectar cualquier posible impacto de procesamiento gradual después de que se completa una implementación y antes de continuar con la siguiente etapa.

Para calcular la cantidad de tiempo que invertimos en procesar una implementación, necesitamos equilibrar el riesgo de causar un impacto mayor si promovemos cambios a varias regiones demasiado rápido, en contraste con la velocidad a la que podemos entregar los cambios a los clientes a nivel global. Hemos descubierto que una buena forma de equilibrar estos riesgos es que las primeras oleadas en la canalización cuenten con mayor tiempo de procesamiento mientras se refuerza la seguridad del cambio, y que las oleadas posteriores cuenten con un tiempo de procesamiento más corto. Nuestro objetivo es minimizar el riesgo de un impacto que afecte varias regiones. La mayor parte de las implementaciones no se observan de forma activa por un miembro del equipo, por lo que los tiempos normales de procesamiento predeterminados de la canalización común son moderados y se verá un cambio en todas las regiones dentro de los cuatro o cinco días laborables. Los servicios que son más extensos o críticos conllevan tiempos de procesamiento y tiempos destinados a sus canalizaciones aún más moderados para que implementen un cambio de forma global.

Una canalización normal espera al menos una hora luego de cada etapa de caja individual, al menos 12 horas luego de la primera oleada regional y al menos dos o cuatro horas después de cada una de las oleadas regionales restantes. Esto sumado al tiempo de procesamiento adicional para las regiones, las zonas de disponibilidad y las celdas individuales en cada oleada. El tiempo de procesamiento

incluye requisitos de espera de un número específico de puntos de datos en las métricas del equipo (por ejemplo, “esperar al menos 100 solicitudes de creación de la API”) para asegurar que se han enviado suficientes solicitudes para sea probable que el código nuevo se haya ejecutado por completo. Durante todo el tiempo de procesamiento, la implementación se restaura de forma automática si se activa la alarma agrupada de nivel alto de gravedad del equipo.

Aunque sucede con poca frecuencia, en algunos casos, un cambio urgente (por ejemplo, un ajuste de seguridad o una mitigación de un evento a gran escala que afecta la disponibilidad del servicio) puede necesitar ser entregado a los clientes más rápido de lo que el tiempo de la canalización suele tomar para procesar los cambios e implementarlos. En estos casos, podemos disminuir el tiempo de procesamiento de la canalización para acelerar la implementación, pero se requerirá un nivel alto de análisis del cambio para poder hacerlo. Para estos casos, necesitamos el análisis de los ingenieros principales de la organización. El equipo debe revisar el cambio de código, así como su urgencia y su riesgo de impacto, con desarrolladores experimentados que tengan un gran conocimiento en seguridad operativa. El cambio aún recorre los mismos pasos en la canalización de la forma usual, pero se dirige a la próxima etapa con mayor rapidez. Administramos el riesgo de una implementación más rápida mediante la limitación de los cambios que se producen en la canalización durante este tiempo para permitir solo los mínimos cambios de código requeridos para solucionar el problema actual, observando las implementaciones de forma activa.

Alarma y bloqueadores de periodos

La canalización evita las implementaciones automáticas en la producción cuando hay un mayor riesgo de causar un impacto negativo. La canalización utiliza un conjunto de “bloqueadores” que evalúa los riesgos de implementación. Por ejemplo, la implementación automática de un nuevo cambio en la producción cuando se está produciendo un problema en el entorno podría producir un impacto peor o más prolongado. Antes de iniciar una nueva implementación en cualquier etapa de la producción, la canalización revisa la alarma agrupada de nivel alto de gravedad del equipo para determinar si hay problemas activos. Si la alarma se encuentra actualmente en estado de alarma, la canalización evita que el cambio avance. Las canalizaciones también pueden comprobar alarmas para toda la organización. Por ejemplo, una alarma a gran escala que indica si hay un gran impacto en los sistemas de otros equipos y evita que se inicie una nueva implementación que pueda sumarse al impacto total. Estos bloqueadores de implementaciones pueden ser anulados por los desarrolladores cuando se necesita implementar un cambio en la producción para recuperarse de un problema de nivel alto de gravedad.

La canalización también se configura con un conjunto de periodos que definen cuando permitir el inicio de una implementación. Cuando se configuran los periodos, se necesita equilibrar dos causas de riesgos en la implementación. Por un lado, los periodos muy cortos pueden causar que los cambios se acumulen en la canalización mientras el periodo está inactivo, lo que aumenta la probabilidad de que cualquiera de esos cambios en la próxima implementación tenga un impacto cuando se active el periodo. Por otro lado, los periodos extensos que se mantengan activos más allá del horario laborable regular incrementan el riesgo de prolongación del impacto de una implementación con error. Fuera de las horas laborables, toma más tiempo que se involucre el ingeniero de turno que durante el día, cuando él y otros miembros del equipo están trabajando. Durante el horario laborable regular, el equipo puede involucrarse más rápido luego de una implementación fallida, en caso de que se necesite una recuperación manual.

La mayor parte de las implementaciones no son supervisadas de forma activa por un miembro del equipo. Por lo tanto, se optimiza el ritmo de las implementaciones para minimizar el tiempo que lleva emplear a un ingeniero de turno, en caso de que se requiera de una acción manual para la recuperación luego de una restauración automática. Suele haber una mayor demora de respuesta de los ingenieros de turno por la noche, en los días feriados y durante los fines de semana, por lo que estos momentos se excluyen de los periodos. En función de los patrones de uso del servicio, es posible que algunos problemas no aparezcan durante horas después de la implementación, por lo que muchos equipos también excluyen de los periodos las implementaciones durante las tardes y los viernes, a fin de reducir el riesgo de necesitar la presencia de ingenieros de turno por la noche o durante el fin de semana luego de una implementación. Hemos descubierto que este conjunto de periodos permite una recuperación rápida incluso cuando se necesita una acción manual, asegura menor cantidad de trabajo para ingenieros de turno fuera del horario regular de trabajo y asegura que se agrupe una pequeña cantidad de cambios mientras los periodos están cerrados.

Canalizaciones como código

El equipo de servicio de AWS habitual posee muchas canalizaciones para implementar distintos microservicios del equipo y tipos de fuentes (código de aplicaciones, código de infraestructura, parches de sistema operativo, etc.). Cada canalización tiene muchas etapas de implementación para un número de regiones y zonas de disponibilidad cada vez mayor. Esto se traduce en una larga configuración para que el equipo administre en el sistema de canalización, in el sistema de implementación y en el sistema de alarma. Además, requiere de un esfuerzo considerable para mantenerse actualizado con las prácticas recomendadas y con nuevas regiones y zonas de disponibilidad. En los últimos años, hemos incorporado la práctica de “canalizaciones como código”, como una manera de configurar canalizaciones seguras y actualizadas de forma fácil y consistente, mediante el modelado en código de esta configuración. Nuestras canalizaciones internas como herramientas de código se extraen de una lista centralizada de regiones y zonas de disponibilidad para agregar fácilmente nuevas regiones y zonas de disponibilidad a las canalizaciones en AWS. La herramienta también permite a los equipos modelar las canalizaciones mediante el uso de la herencia, el ajuste de una configuración común en las canalizaciones del equipo (por ejemplo, qué regiones pertenecen a cada oleada y cuán extenso debe ser el tiempo de procesamiento para cada oleada) y el ajuste de la configuración de las canalizaciones de todos los microservicios como una subclase que hereda toda la configuración común.

Conclusión

En Amazon, hemos creado nuestras prácticas de implementación automatizada de manera gradual y de acuerdo con lo que nos permite generar equilibrio entre la seguridad y la velocidad de la implementación. Al mismo tiempo, queremos minimizar la cantidad de tiempo que los desarrolladores deben pasar preocupándose por las implementaciones. Reforzar la seguridad de las implementaciones automatizadas en el proceso de lanzamiento mediante la utilización de pruebas de preproducción extensas, restauraciones automáticas e implementaciones de producción escalonadas nos permite minimizar el impacto potencial en la producción causada por las implementaciones. Esto significa que los desarrolladores no necesitan observar las implementaciones en la producción de forma activa.

Con canalizaciones completamente automatizadas, los desarrolladores utilizan revisiones de código para verificar que su código esté bien y también para confirmar que el cambio esté listo para incluirlo en la producción. Una vez que el cambio se incluye en el repositorio de código fuente, el desarrollador puede

continuar con la próxima tarea y olvidarse de la implementación. Puede confiar en que la canalización insertará el cambio en la producción de forma segura y cautelosa. La canalización automatizada se encarga de realizar implementaciones en la producción de forma continua varias veces al día, al mismo tiempo que mantiene el equilibrio entre la seguridad y la velocidad. Al modelar nuestra práctica de entrega continua de código, es más sencillo que nunca para los equipos del servicio de AWS configurar las canalizaciones para implementar los cambios de código de manera automática y segura.