

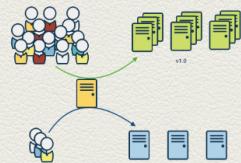
Cuatro principios para Releases de software de bajo riesgo

Integrantes

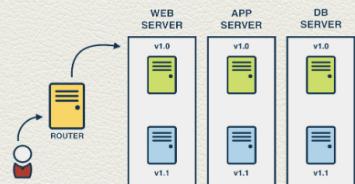
Cattaneo Daniel
Malisani Asis Maria Sol
Simes Pellegrini Valter
Turra Matias

#1 Los releases de bajo riesgo son incrementales

Usando este patrón desplegamos la nueva versión de la aplicación acompañándonos de la vieja versión. Esta implementación ofrece una forma rápida de retroceder: si algo sale mal, se vuelve a conectar el router en el entorno azul.

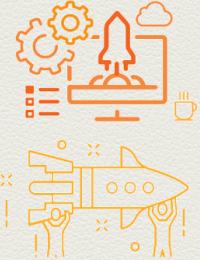


Se implementa el cambio en un pequeño subconjunto de usuarios y a medida que se adquiere más confianza en la nueva versión, se puede enrutar a más usuarios. Cuando todos los usuarios se han enrutado a la nueva versión, se puede desactivar la infraestructura anterior. Si se encuentra algún problema con la nueva versión, se puede redirigir a los usuarios a la versión anterior.



Canary Releasing

Los conceptos de deploy y release son muy frecuentemente utilizados como una unidad y no se explica hacer uno si el otro, pero es conveniente hacer la diferencia entre uno y otro y utilizarla a nuestro favor. Deploy consiste en la instalación de una versión específica de tu software en un ambiente en particular. En cambio, el Release es el momento en que se hace disponible para usuarios un nuevo software o una parte de este.



#2 Desacoplar Despliegue de Release

Separar estos dos momentos nos da muchas ventajas a la hora de la implementación de nuevas funcionalidades. Algunas son:

- Hacer testing a las nuevas funcionalidades en el entorno en que van a ser usadas. Esto hace que terminado el testing, el release sea mucho más fácil de realizar y el software haya sido probado en el ambiente en que va a ser ejecutado.
- Permite sectorizar los Releases y de esta forma recibir feedback específico según las características demográficas de cada grupo.
- Probar una funcionalidad en un grupo reducido de usuarios para luego ir escalando poco a poco la cantidad, de esta forma se evitan saturaciones en los servidores y en el caso de que haya un defecto puede ser descubierto tempranamente y este no afecta a la totalidad de los usuarios.

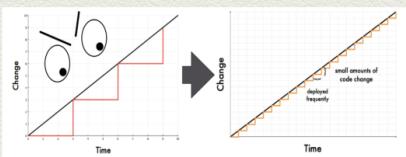
#3 Concentrarse en reducir el tamaño del lote

Es una de las técnicas más poderosas para promover el flujo de características desde los "cerebros" a los usuarios. Al reducir el tamaño del lote se puede desplegar más frecuentemente, pero ¿Cómo esto reduce el riesgo?

Cuando por ejemplo se despliega grandes cantidades de trabajo, desplegar nuevamente puede generar mucho dolor. La solución es desplegar frecuentemente y hacerle frente a este dolor de a pequeñas cuotas se podría decir.

Se reduce el riesgo del despliegue de cada lote individual por tres razones:

- Al ser frecuente el despliegue, también lo es el proceso de despliegue. Así se detectan y arreglan fallas más rápido. El proceso a su vez cambiará poco en cada despliegue.
- Descubrir qué es lo que salió mal es mucho más simple si los lanzamientos son frecuentes. Es así que los cambios son pequeños y es allí un buen lugar para empezar a buscar.
- Retroceder en un cambio pequeño es mucho más fácil, es menor la cantidad de componentes afectados y a nivel equipo es mejor persuadir el retroceso en cambios pequeños.



#4 Optimizar para la resiliencia

Hay dos acercamientos fundamentales diseñando un sistema. Se puede minimizar el mientras tanto entre fallos (MTBF) o el mientras tanto para restaurar el servicio (MTRS). Un BMW está optimizado para MTBF, porque encontrar un error es raro pero si pasa, va a costar mucho. Mientras que por ejemplo un Jeep, optimizado para MTRS, se rompe básicamente todo el tiempo pero es posible desarmarlo y armarlo en poco tiempo.



mean time to restore service (MTRS)

mean time between failures (MTBF)

La habilidad para restaurar tu sistema a un estado de línea base en un tiempo estimable es vital no solo cuando el despliegue fracasa, sino también como parte de tu estrategia de recuperación post desastre.



El mayor enemigo de crear un sistema resiliente es lo que se conoce como "piezas de arte": componentes de tu sistema que han sido desarrollados a lo largo de años y que si fallan, sería imposible reproducirlos en un tiempo predecible. Se debe encontrar una forma de crear una copia de ese componente para propósitos de testing y para poder crear una nueva instancia de ellos en ocasiones de desastre.

La parte más importante de crear sistemas resilientes es el humano. Cuando un servicio se cae, es imperativo que todos sepan que proceso seguir para diagnosticar el sistema y levantarla y que corra de nuevo, y también que todos los roles y habilidades necesarias para realizar estas tareas estén disponibles y puedan trabajar en conjunto. Entrenamiento y colaboración efectiva son las claves.

