Escuela Politécnica Nacional

Construcción y evolución de software

Nombres: Sebastián Ramos, Juan Mateo Quisilema

Caso de estudio: GitLab – Incidente de pérdida de datos.

Resumen del caso.

El 31 de enero de 2017 GitLab.com sufrió un grave incidente en uno de sus servidores de base de datos que provocó la eliminación accidental de datos de producción. El servicio estuvo parcial o totalmente degradado durante horas y se perdió información generada en un periodo de aproximadamente 6 horas.

Mientras intentaban resolver problemas de replicación entre la base de datos primaria y la réplica, un ingeniero ejecutó por error un proceso de resync (eliminación y reconstrucción) sobre la base primaria en lugar de la réplica. En cuestión de segundos se eliminaron cientos de gigabytes de datos y parte de esa información no pudo recuperarse. La interrupción duró muchas horas mientras se restauraba desde copias de seguridad y se evaluaba la pérdida.

2) Clasificación del mantenimiento

Tipo principal: Correctivo.

Fue principalmente correctivo porque el objetivo inmediato fue reparar un daño concreto en producción: recuperar servicio y restaurar los datos perdidos.

Contenía además elementos preventivos y perfectivos: tras el incidente, GitLab implementó mejoras en procedimientos y automatizaciones, fortaleció backups y mejoró la documentación de los procesos para evitar recurrencias.

No es principalmente adaptativo, ya que no se trató de adaptar software a un nuevo entorno.

3) Procesos SCM involucrados

Control de versiones

Repositorios de código: los scripts usados para sincronización y los procedimientos de runbook deberían estar versionados en Git. Esto permite revisar cambios recientes que podrían haber contribuido al error.

Branches y despliegues: para cambios en el código de operación se habrían creado ramas y revisiones antes de desplegar en producción.

Gestión de Configuración y cambios

Rama de emergicia/rollback: al detectar el problema, el equipo probablemente activó procedimientos de emergencia para coordinar restauración desde backups. En entornos con buen SCM, se crea una rama/etiqueta (tag) de la versión estable conocida como punto de restauración.

Aprobaciones urgentes: el parque de restauración requirió decisiones y autorizaciones rápidas, con registros de quién aprobó cada paso.

Gestión de Releases

El despliegue fue manual y de emergencia, se restauraron snapshots y backups en un entorno controlado antes de volver a producción.

Gracias al control de versiones y registros de cambios se pudo trazar qué comandos y scripts se ejecutaron y quién los ejecutó

4) Impacto en el ciclo de vida del software (SDLC)

Operaciones/Despliegue: El impacto fue inmediato en producción; gran parte del esfuerzo se centró en restauración y despliegue controlado.

Pruebas: Tras la restauración hubo que validar la consistencia de datos y posiblemente ejecutar pruebas de regresión sobre funcionalidades críticas para asegurarse de que la restauración no introdujo nuevas inconsistencias.

Planificación: El incidente obligó a re-planificar tareas y priorizar mejoras de resiliencia y backups antes que nuevas features.

Desarrollo: Revisiones de scripts operativos y automatizaciones; se implementaron cambios en el código y runbooks.

5) Beneficios del SCM en este caso.

Trazabilidad y auditoría: Al tener scripts, runbooks y cambios en control de versiones se pudo identificar qué se ejecutó y cuándo.

Capacidad de Reversión y Puntos de Restauración: Versionado de procedimientos y tags permite recrear el estado conocido y comparar con versiones previas.

Coordinación y Comunicación: Un buen SCM (con issues y pipelines) facilita coordinar las tareas de emergencia, asignar responsabilidades y documentar cada acción durante la recuperación.

Reducción de Errores Futuros: Con SCM se pueden automatizar pruebas y deploys, reduciendo la dependencia de comandos manuales peligrosos.

6) Recomendaciones (lecciones aprendidas aplicables)

1. **Probar backups regularmente** (restore drills) para asegurar que las copias realmente permiten recuperar todos los componentes del servicio.
2. **Separar ambientes y permisos**: minimizar la posibilidad de ejecutar scripts sobre producción desde servidores de administración mal identificados.
3. **Hardening de comandos peligrosos**: añadir confirmaciones extra, prompts obvios (colores), y restricciones de rol para comandos destructivos.
4. **Versionar runbooks y scripts operativos** y exigir revisiones de pares antes de cambios que afecten a producción.
5. **Automatizar tareas críticas** con pipelines validados en vez de permitir ejecuciones manuales inseguras.
6. **Documentar y practicar procedimientos de incidente**, incluyendo comunicación externa para usuarios afectados.