ERNI

DevOps 101: Consejos y mejores prácticas de DevOps

3 SISTEMA DE CONTROL DE VERSIONES

4 INTEGRACIÓN Y DESPLIEGUES CONTINUOS

5 AUTOMATIZACIÓN DE LA INFRAESTRUCTURA

6 MÉTRICAS Y MONITORIZACIÓN

7 AUDITORÍA DE SEGURIDAD DE LA INFRAESTRUCTURA

8 PLAN DE GESTIÓN DE LA CONFIGURACIÓN

9 DOCKERIZA TUS SERVICIOS

10 CONTACTA CON NUESTRO EQUIPO DE DEVOPS

SISTEMA DE CONTROL DE VERSIONES

Empieza a usar Git como sistema de control de versiones. A diferencia de los sistemas de control de versiones centralizados, las ramas de Git son sencillas de crear y fáciles de combinar. Esto facilita el flujo de trabajo de las *feature branches*, popular entre muchos usuarios de Git. Las *feature branches* proporcionan un entorno aislado para cada cambio en su código fuente. Cuando un desarrollador quiere empezar a trabajar en algo, sea grande o pequeño, crea una nueva rama. Esto asegura que la rama master siempre contiene código listo para producción.

Usando las *feature branches* no solo es más seguro que editar directamente el código de producción, sino que también proporciona beneficios organizacionales. Te permiten representar el trabajo de desarrollo con la misma granularidad que su backlog de agile.

Usa *Pull requests*. Una *Pull request* es una forma de pedirle a otro desarrollador que fusione una de sus ramas en su repositorio. Esto no solo facilita que los líderes de proyectos realicen un seguimiento de los cambios, sino que también les permite a los desarrolladores iniciar discusiones sobre su trabajo, antes de integrarlo con el resto del código fuente.

Define una estrategia de ramas. Git hace que sea más fácil que nunca crear ramas y fusionarlas. Una posible consecuencia es que los desarrolladores tiendan a crear muchas más ramas. No lo harán necesariamente, pero es altamente probable que ocurra. Llegado al caso, es necesario documentar, educar y garantizar que todas las personas del mismo equipo trabajen de la misma manera.

Si deseas lograr un escenario de despliegue continuo, considera usar una estrategia de *Trunk Based Development*, que tiene una única rama de larga duración, y las *feature branches* se fusionan directamente en ella.

Define el control de versiones de tu aplicación. Lo más probable es que, a medida que la aplicación crezca, la cantidad de dependencias incluidas en el proyecto también crezca con ella. Cuantas más dependencias, más difícil será su gestión. A veces, es posible que te resulte difícil actualizar una dependencia sin romper nada.

Otras veces, las dependencias son demasiado flexibles. Algunas dependencias ofrecen soporte para el futuro, más de lo que se requiere a veces. Lo que comúnmente se conoce como “dependency hell”.

Una solución sencilla para esto es empezar a versionar correctamente, con un conjunto común de reglas, que ayuda a que cada dependencia permanezca en el mismo estado tanto como sea posible.

INTEGRACIÓN Y DESPLIEGUES CONTINUOS

Empieza a automatizar tus despliegues utilizando Pipelines. Para obtener un feedback rápido, necesitas que la ejecución de los pipelines dure el menor tiempo posible. Los pipelines permiten dirigir los cambios a través de distintos ciclos de test automatizados, distribuyendo los cambios a entornos de Staging y después a producción. Cuanto más exhaustivos sean tus pipelines, mayor seguridad tendrás de que los cambios que introduzcas no tengan efectos no deseados en el despliegue a producción.

Desde una perspectiva de seguridad operacional, tu sistema de CI/CD representa uno de los puntos más críticos de tu infraestructura a proteger. Dado que tu sistema de CI/CD tiene acceso completo al código fuente y credenciales de varios entornos, es esencial asegurar la integridad de los datos del sistema y garantizar la integridad de tu producto. Es importante aislar tu sistema de CI/CD lo mejor posible.

Asegúrate de que la única forma que tengas de desplegar a producción sea a través de tu herramienta de CI/CD. Los errores en los pipelines se ven de forma inmediata y te permiten bloquear la promoción de tus cambios a entornos posteriores cuando se detecta un error. Es un gran mecanismo de control que protege los entornos más críticos de código con errores.

Mantén paridad en tus entornos con producción siempre que sea posible. Algunas diferencias entre staging y producción pueden tener lugar a nivel de recursos, pero asegúrate de que esas diferencias estén bien controladas. Diferencias significativas entre los entornos puede dar lugar a comportamientos inesperados y errores que no se han podido detectar previamente. Cuanto más distintos sean tus entornos de producción, más difícil será testear y encontrar errores proactivamente antes de desplegar en producción, invalidando los resultados de los tests.

Construye tu código una sola vez y distribúyelo con el pipeline. Esto te permite prevenir problemas que surjan cuando el código se compila o empaqueta muchas veces, pudiendo dar lugar a inconsistencias que vayan a parar a los artefactos generados. Construir tu código de forma individual para cada entorno significa que los tests en entornos previos no se ejecutan sobre lo mismo que se despliega después, invalidando los resultados.

Lanza tus tests de forma local antes de subirlos a tu repositorio de código fuente. Los desarrolladores deberían asegurarse de realizar el máximo número de pruebas posibles antes de subir los cambios al repositorio. Esto hace posible detectar cambios con errores que puedan bloquear a otros miembros del equipo.

Lanza los tests en entornos efímeros siempre que sea posible. A menudo, es una buena práctica lanzar los tests en contenedores para abstraerse de las diferencias entre los sistemas y para proporcionar APIs estándar que unan componentes a distintas escalas. Dado que los contenedores pueden correr con pocos recursos, los efectos residuales de los tests no se heredan por las siguientes ejecuciones, y por lo tanto se elimina la posibilidad de que los resultados posteriores se puedan ver contaminados.

AUTOMATIZACIÓN DE LA INFRAESTRUCTURA

Todas las especificaciones de la infraestructura deben estar explícitamente codificadas en templates, usando herramientas estándar de IaC (Infraestructura como código). Esos templates serán la única fuente de configuración de infraestructura y describirán de forma concreta y exacta qué componentes de Cloud se utilizan, cómo están relacionados y cómo los distintos entornos están configurados.

Los templates serán la propia documentación. No debería haber muchas más instrucciones adicionales que el departamento de IT deba ejecutar. La única información adicional que tendría cabida son diagramas o instrucciones para realizar el setup o despliegue de los templates.

Almacena tus templates en un sistema de control de versiones. De este modo, cualquier cambio que tenga lugar en tu infraestructura tendrá la trazabilidad de un sistema transaccional como puede ser *git*. De este modo, puedes incluso probar tu infraestructura antes de desplegarla.

Existen enfoques como *GitOps*, orientados a orquestadores como *Kubernetes*, en los que los propios sistemas se encargan de actualizar o hacer *rollbacks* en base a los templates del repositorio.

Incluye tus templates dentro del flujo de CI/CD. El testing debería aplicarse rigurosamente a las configuraciones de infraestructura determinadas que estés utilizando, para asegurar que no hay problemas después del despliegue. En función de tus necesidades, una lista de tipos de test – unitarios, regresión, integración, etc. – deberían ejecutarse durante esta fase. Los tests automáticos pueden lanzarse cada vez que se produzca un cambio en tus templates.

Haz que tu infraestructura sea modular e inmutable. La infraestructura modular limita la cantidad de cambios que pueden hacerse en tus templates. Cuanto más pequeños sean los cambios, más sencillo será detectar errores de forma ágil. Hacer tu infraestructura inmutable proporciona consistencia, evita cambios de configuración no deseados y restringe el impacto de cambios no documentados o trazados en tu infraestructura. También mejora la seguridad y hace que la resolución de problemas se simplifique y aísle a un escenario controlado.

MÉTRICAS Y MONITORIZACIÓN

Un tipo de métrica que las organizaciones suelen obviar son las métricas de procesos DevOps. Una vez se define qué es DevOps dentro de la organización, puedes identificar los distintos retos y medir su eficiencia.

Dado que la finalidad de DevOps se centra en una entrega continua y despliegue de código de la forma más ágil posible, hay métricas muy relevantes que tu organización debería monitorizar. ¿El objetivo? Mejorar la velocidad, el rendimiento y la calidad.

Métricas como la frecuencia de despliegue, la cantidad de nuevas *features* o cambios, el tiempo de despliegue y el tiempo que se tarda desde que se define una *feature* hasta que se aplica en producción, permiten medir la velocidad de tus equipos.

Algunas otras métricas como *tickets* de clientes, porcentaje de tests automáticos pasados, ratio de errores detectados antes de producción, despliegues fallidos y número de errores permiten medir la calidad del servicio que se está dando.

Uso de la aplicación, tráfico que soporta y rendimiento de la aplicación permiten medir la capacidad de tu servicio.

El tiempo medio de detección y tiempo medio de recuperación permiten medir la resiliencia del servicio que proporcionas a tus clientes.

Dicho todo esto y con toda esta información, se pueden crear modelos predictivos que puedan prevenir caídas de servicio o red, degradaciones de servicio y/o recursos que se van a necesitar en momentos concretos.

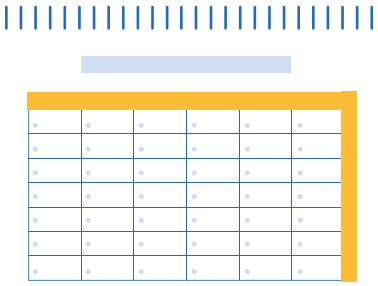
AUDITORÍA DE SEGURIDAD DE LA INFRAESTRUCTURA

Introduce una herramienta que capture requisitos de seguridad y pueda aplicar distintos perfiles sobre tus sistemas. Algunos ejemplos de elementos *testeables* son paquetes, servicios, usuarios y/o sistemas o ficheros de sistemas. Estas herramientas de auditoría incluyen numerosos recursos para sistemas Linux o Windows, ofreciéndote lo que tus sistemas necesitan para ser seguros.

Estas herramientas no sólo son herramientas de monitorización, sino que también permiten aplicar el *hardening* o acciones necesarias siempre que se detecte alguna vulnerabilidad en cualquiera de los componentes.

Incluye la herramienta dentro de tu flujo de CI/CD. Cada vez que se despliega un nuevo sistema, deberá monitorizarse por la herramienta. El uso conjunto de esta herramienta con la infraestructura como código, proporciona un alto nivel de automatización y *compliance*, permitiendo que tus equipos puedan enfocarse en tareas de prevención en vez de solucionarlas de manera reactiva.

PLAN DE GESTIÓN DE LA CONFIGURACIÓN



Las herramientas de gestión de configuración te previenen de hacer pequeños o grandes cambios sin documentar y/o trackear. Sin ellos, puedes terminar sin saber qué hay en un servidor, o qué actualizaciones de software han tenido lugar, haciéndolo difícil de mantener. Con las herramientas de gestión de configuración, puedes replicar de manera exacta un entorno con las configuraciones adecuadas y el software necesario porque sabes de forma explícita qué existe en el entorno original.

El rol de las herramientas de gestión de configuración es permitir un aprovisionamiento simple y rápido, así como mantener un estado determinado del sistema. Tradicionalmente, estas tareas se hacían de forma manual o con scripts generados por los propios administradores de sistemas. El uso de herramientas estándar proporciona un alto nivel de automatización, reducción de costes, reducción de complejidad y por supuesto, de errores.

Mediante la automatización, las herramientas de gestión de configuración pueden provisionar un nuevo servidor o servicio en cuestión de minutos, con probabilidades muy bajas de error. Puedes utilizar estas herramientas para volver un servidor a un estado determinado, sin la necesidad de los scripts anteriormente mencionados y sin la necesidad de tener que restaurar copias de seguridad.

DOCKERIZA TUS SERVICIOS

Utilizar una aproximación enfocada a contenedores en tus equipos de desarrollo proporciona muchos beneficios. Permite establecer ciclos de entrega de software mucho más rápidos, proporciona portabilidad a las aplicaciones y las prepara para arquitecturas enfocadas a microservicios o entornos de cloud *serverless*. ¿Cómo empezar?

Lo primero en lo que se necesita pensar es en procesos *stateless*, instancias desechables y manejo de logs como eventos. Necesitarás empezar a analizar las dependencias de tu aplicación y desestructurarla.

Después necesitarás encontrar que imagen base se adapta mejor a tu aplicación. Piensa en características como el tamaño, las capas, los tags e imágenes base comunes. Al final, la imagen es un paquete que también necesita ser securizado y tiene que poder distribuirse de forma sencilla.

Organiza y gestiona la configuración, como los niveles de logs y dónde se encuentran, las credenciales y configuraciones o *settings* de tu aplicación. Éstas no son responsabilidades de los contenedores, y la mejor manera de gestionar esta información/configuración es mediante variables de entorno.

Dada la naturaleza volátil de los contenedores, tiene que existir una capa extra sobre los mismos que permita tenerlos bajo control. Es donde aparece la necesidad de usar orquestadores, descubrimiento de servicios y gestión de la configuración. Te permitirán una gran flexibilidad para escalar, trackear y controlar dinámicamente tus servicios.

Por último y no menos importante, hay que tener en cuenta la seguridad. Integra en tus sistemas de CI/CD herramientas de *hardening* que te aseguren que tus contenedores están correctamente aislados, que los privilegios están correctamente delimitados y que los recursos también están limitados.

Contacta con nuestro equipo de DevOps

Spain: +34 93 667 77 76
mailto: info@erni-espana.es
www.betterask.erni

SWITZERLAND I GERMANY I SINGAPORE I PHILIPPINES