Engineer Plataformas

Automatización y CI/CD

61 logros

Conecte las partes de su sistema

Los trabajos reunidos en «Automatización y CI/CD» muestran una parte de un sistema en funcionamiento. Podemos ayudar a conectar esa parte con el resto de su operación. Monitorización y alertas que avisan cuando un fallo aún es pequeño, y cifras que muestran con qué frecuencia se mantiene así.

Explore nuestros servicios Póngase en contacto →

Encuentre un reto comparable

Siga un ejemplo que se parezca a su situación. Su reseña describe las decisiones detrás del resultado, más allá de la etiqueta tecnológica. Primer panel de observabilidad de la empresa — la primera visión de la empresa sobre sus propios sistemas

Primer panel de observabilidad de la empresa Póngase en contacto →

Primer panel de observabilidad de la empresa

Desarrollé y puse en marcha el primer panel de observabilidad de la empresa, con información del rendimiento del sistema en tiempo real y visualizaciones de datos en el televisor grande de la oficina.

10 servidores PostgreSQL y MS SQL en Ubuntu

Diseñé, desplegué y mantuve 10 servidores PostgreSQL y MS SQL en VPS con Ubuntu Linux, para mantener un rendimiento y una fiabilidad adecuados.

1.000 reglas RBAC para servidores Linux

Reforcé la seguridad de los datos mediante 1.000 reglas RBAC para desarrolladores, instancias de aplicaciones, PostgreSQL, MS SQL y otros servidores Linux, evitando accesos no autorizados y documentando la configuración mediante automatización con Ansible.

Despliegue GIS SaaS con GitHub Actions

Automaticé el despliegue de una aplicación GIS SaaS, el procesamiento de datos y el sistema de informes con GitHub Actions CI/CD, Python, Bash y SQL.

Entrega automática de 20 flujos de datos GIS

Automaticé la entrega de 20 flujos de datos GIS y procesos ETL de datos de aplicaciones, simplificando la automatización de la infraestructura y la elaboración de informes.

100 copias críticas automatizadas con Barman

Automaticé 100 copias de seguridad de datos críticos con Barman, Google Cloud, Bash y Python, preservando la integridad de los datos entre bases de datos.

20 aplicaciones Docker en Google Cloud y AWS

Desplegué y mantuve 20 aplicaciones en contenedores Docker, resolví incidencias con Podman y Kubernetes y administré aplicaciones basadas en R en Google Cloud y AWS.

Gestión de accesos con M365 y SSO

Prevení brechas de seguridad al dirigir iniciativas de gestión de accesos con M365, 1Password, Red Hat SSO y OKTA SSO.

Menor riesgo con Grafana y Prometheus

Reduje los riesgos operativos mediante un panel de monitorización con Grafana y Prometheus, mejorando la fiabilidad del sistema.

Mejor comunicación y colaboración del equipo

Mejoré la comunicación y la colaboración del equipo mediante Slack, Mattermost, 1Password y Jira, ahorrando 8.000 horas de trabajo.

Procesos renovados: 8.000 horas ahorradas

Reorganicé los procesos internos y ahorré 8.000 horas mediante mejoras en la arquitectura del software, los sistemas y la planificación.

Administración de infraestructura de red

Administré la infraestructura de red de más de 1.000 servidores, atendiendo al despliegue, la seguridad y la resolución de incidencias de los sistemas.

SSL/TLS automático para 100 aplicaciones Docker

Automaticé la creación de certificados SSL/TLS para 100 aplicaciones Docker, manteniendo conexiones seguras entre servidores Ubuntu Linux.

CI/CD optimizado: 4.000 horas ahorradas

Simplifiqué los procesos CI/CD y ahorré 4.000 horas mediante la automatización de los flujos de desarrollo de software.

4.000 horas ahorradas con CI/CD GitHub y GitLab

Simplifiqué los procesos de análisis de datos y desarrollo de software y ahorré 4.000 horas mediante prácticas CI/CD con GitHub, GitLab, Bash y Python.

Datos automatizados con Python y PL/pgSQL

Automaticé tareas de procesamiento de datos con scripts de Shell, PL/pgSQL, Python y Transact-SQL, mejorando la productividad y la eficiencia.

Contrato de API generado desde la base de datos

Generé el contrato de la API desde la base de datos hacia las capas externas — OpenAPI, un cliente TypeScript tipado de 44.076 líneas, 61 manejadores simulados y los límites que aplica la interfaz — con una comprobación en cada paso que falla ante cualquier divergencia.

Infraestructura Azure como código

Aprovisioné infraestructura Azure como código con Bicep — Container Apps, PostgreSQL Flexible Server, Front Door/WAF y redes — en los entornos de desarrollo, preproducción y producción.

Flujos CI/CD en GitHub Actions

Desarrollé flujos CI/CD en GitHub Actions con una imagen de producción distroless para la interfaz y promoción entre varios entornos.

578 tareas de automatización go-task

Redacté 578 tareas de automatización go-task para los modos de desarrollo nativo, Docker y HTTPS, así como para análisis estático, pruebas, bases de datos y despliegue.

Despliegues completos de la plataforma en Azure

Asumí los despliegues completos de la plataforma en Azure y gestioné las versiones en desarrollo, preproducción y producción.

Cero advertencias en seis lenguajes

Fijé un criterio de calidad de cero advertencias en seis lenguajes — Go, TypeScript, SQL, Python, Shell y Markdown — aplicado mediante hooks de pre-commit.

Colaboración del equipo automatizada

Automaticé la colaboración del equipo, la gestión de contraseñas, tareas y tiempo, y desarrollé un sistema semiautomático de presentación de proyectos que elevó la productividad del equipo.

Gestor de contraseñas para toda la empresa

Integré un sistema de gestión de contraseñas para toda la empresa, reforzando la seguridad y simplificando el control de accesos.

Pruebas para Go, SQL y navegador

Desarrollé una batería de pruebas automatizadas por capas — 981 pruebas Go, 543 especificaciones de interfaz y navegador y 494 pruebas de comportamiento SQL — con pruebas de mutación, pruebas basadas en propiedades y un control obligatorio de accesibilidad.

Motor de 268 controles para el repositorio

Desarrollé el motor de comprobaciones del repositorio — 268 controles de commit registrados, 277 reglas de análisis estático y 15 reglas ESLint propias —, además de 146 pruebas de los propios controles, para que la compilación haga cumplir el criterio de calidad.

Trabajo lento trasladado a una cola River

Saqué las tareas lentas del procesamiento de solicitudes y las trasladé a una cola River — 15 módulos de trabajadores, 8 tareas programadas y 20 tareas pg_cron — para responder a las solicitudes mientras el trabajo continúa en segundo plano.

Análisis continuo de seguridad

Establecí un programa continuo de seguridad — análisis de código, DAST, comprobación de dependencias y vulnerabilidades, generación de SBOM, detección de secretos y acciones fijadas por SHA —, junto con 21 auditorías de seguridad documentadas.

Infraestructura definida en un repositorio

Desarrollé la infraestructura de la empresa mediante 19 playbooks de Ansible y 34 roles en 12.065 líneas de YAML, haciendo converger un servidor activo hacia un estado declarado con todas las ejecuciones idempotentes.

SSH reforzado con 24 directivas verificadas

Reforcé SSH con 24 directivas verificadas mediante validación en tres etapas — archivo candidato, configuración ensamblada y lectura de la configuración por el propio demonio —, después de que esta última detectara que el servidor en ejecución anulaba silenciosamente dos de las veinticuatro.

Bloqueo de intrusiones probado en cada ejecución

Verifiqué de extremo a extremo el bloqueo de intrusiones en cada ejecución de refuerzo, bloqueando una dirección reservada para pruebas, comprobando la regla resultante del núcleo y retirando el bloqueo en un bloque de limpieza garantizado, para que una jail averiada haga fallar la ejecución en vez de presentarse como operativa.

Reglas de cortafuegos verificadas por posición

Verifiqué las reglas del cortafuegos por su posición, además de por su existencia, leyendo la lista numerada de reglas y la cadena activa del filtro de paquetes, porque una regla presente puede no recibir nunca un paquete.

Automatización de copias con un método explícito de restauración

Implementé automatización de copias con restic, retención, comprobaciones de integridad y un flujo mensual de prueba de restauración; documenté la activación del almacenamiento y la verificación de restauración real como pasos operativos separados.

Monitorización por ausencia de señal

Desarrollé una monitorización por ausencia de señal que solo avisa de actividad mientras la memoria y el disco están en buen estado, de modo que un servidor degradado genera una alerta al quedar en silencio; además detecté seis variables llamadas «free» cuando el control medía correctamente «available», valores separados por un orden de magnitud en una máquina de 464 MB.

Modo de comprobación Ansible fiel a la realidad

Hice que el modo de comprobación reflejara la realidad en toda la plataforma tras detectar seis sondeos que decidían a partir de un valor que el servidor nunca había proporcionado, porque el módulo command de Ansible informa de éxito con --check aunque omita la orden por completo.

Comprobación de secretos en un segundo

Añadí una comprobación previa que ejecuta en aproximadamente un segundo el mismo código de convergencia sobre secretos locales del operador, después de que una ejecución de producción parcialmente aplicada fallara en su novena tarea con los ajustes de intercambio ya escritos en el servidor activo.

Zona DNS de 20 registros como código

Concilié declarativamente una zona DNS de 20 registros con la API de Cloudflare mediante puntos de entrada separados para auditoría y exportación BIND, y desactivé el proxy CDN por motivos de privacidad después de implementarlo.

Menor exposición en cada unidad systemd

Reduje la exposición de aislamiento systemd en todas las unidades que instala la plataforma — un servicio de monitorización por ausencia de señal pasó de 9,6 UNSAFE a 1,5 y una forja git expuesta a Internet de 8,3 EXPOSED a 1,5 — y añadí una comprobación del analizador durante la convergencia tras encontrar una directiva mal escrita que se ignoraba silenciosamente en tres plantillas de unidades.

7 roles de informes de solo lectura

Desarrollé siete roles de informes de solo lectura que representan un servidor activo en Markdown — inventario técnico, acceso, git, métricas, tráfico, seguridad e inventario de proveedores — bajo la regla de no imprimir ninguna cifra que la ejecución no haya medido.

Su código en su propio servidor

Desplegué la forja git propia de la empresa con Soft Serve, privada por defecto, sin panel web y con el puerto SSH limitado a loopback detrás de un servidor de salto, e hice de su página de entrada un artefacto de compilación del sitio principal en lugar de una copia mantenida a mano.

Contenedores con Podman y Quadlet

Desplegué una plataforma de contenedores con Podman y Quadlet bajo systemd en lugar de Docker, porque la publicación predeterminada de puertos en redes bridge de Docker puede eludir las reglas UFW, y asigné a los despliegues un usuario sin privilegios con una única orden fija, en vez de root.

Segundo servidor aprovisionado desde código

Automaticé el aprovisionamiento de un segundo servidor en un segundo proveedor de nube y creé el cortafuegos antes que la máquina para que naciera protegida, con los cortafuegos de ambos proveedores configurados directamente mediante sus API REST para evitar una colección de terceros.

Regla de alcance tras una fuga de credenciales

Incorporé al repositorio una regla de alcance tras una reorganización que había introducido el inventario, los permisos de cortafuegos y textos de otra empresa, y mantuve los residuos en cuarentena bajo el detector de secretos, sin excluirlos.

Secretos operativos separados por servidor

Separé los cuatro secretos operativos de cada servidor tras constatar que dos servidores que comparten una señal de actividad alertan menos que dos señales independientes, y que una contraseña compartida de copias de seguridad puede exponer las copias de ambos servidores, incluso en repositorios separados, si alguien la obtiene y puede leer los datos cifrados y los archivos de claves de ambos repositorios.

Control de commits con 22 analizadores

Desarrollé un control obligatorio de commits con 22 analizadores de una línea y cinco que requieren una explicación, sin nivel de advertencias ni supresiones en línea permitidas, que cubre HTML, CSS, JavaScript, Python, YAML, Markdown, Shell, enlaces, ortografía, secretos y tipografía.

Una red de seguridad que ya no cuesta media hora

Reduje el tiempo del control de calidad del sitio basado en navegador de 1.636 segundos a 615 al ordenar las comprobaciones de mayor a menor duración y ejecutarlas en un grupo limitado a cuatro trabajadores, tras medir que el orden alfabético requería 320 segundos frente a 224.

Espejos del sitio en Gemini y Gopher

Construí salidas Gemini y Gopher a partir de la misma fuente del sitio web; el logro registra 777 documentos en cada formato en aquella etapa, con cero bytes de HTML modificados al añadir los formatos.

Sitemap corregido: fechas que eran iguales

Corregí un sitemap en el que 172 de 176 URL compartían una misma fecha de modificación, tomando la fecha del historial git tras constatar que la exportación reescribe todos los archivos en cada ejecución.

Control de calidad de 10.242 líneas analizado

Apliqué un formateador y un analizador a 10.242 líneas de JavaScript del control de calidad tras constatar que eran el mayor bloque de código del repositorio y el único que ninguna herramienta revisaba; corregí 13 hallazgos sin suprimir ninguno.

Generador: 981 pruebas y 92 % de cobertura

Mantuve el generador bajo 981 casos de prueba con un mínimo del 92 % de cobertura de ramas y las advertencias tratadas como errores, y verifiqué la idempotencia ejecutando dos veces todo el flujo de construcción desde un archivo vacío y exigiendo que la segunda pasada no cambiara nada.

Cada regla del analizador tratada como error

Activé como errores todas las reglas del analizador Python y resolví 1.815 hallazgos hasta llegar a cero, con un motivo escrito para cada una de las escasas excepciones y dos de ellas respaldadas por una comprobación, además de un comentario.

Pruebas para los propios verificadores

Escribí pruebas para los propios verificadores tras constatar que un verificador que solo recibe entradas correctas puede acabar informando de éxito por no haber leído nada; introduje una falta de ortografía para confirmar su detección y obtuve un intervalo de identificadores de la base de datos en vez de fijarlo en la prueba.

Diferencias detectadas entre hooks y controles

Detecté que los hooks de commit y el control de calidad ejecutaban comprobaciones distintas aunque un documento prometía que eran iguales, al comparar ambas listas en una prueba: las cinco que solo se ejecutaban a mano eran las que revisaban la redacción del CV.

Ensayo del hook CI: 2 defectos detectados

Ensayé el hook CI del servidor de la forja y encontré dos defectos imposibles de alcanzar leyendo el archivo: una alternativa que introducía un argumento irresoluble en la entrada del hook y una variable de entorno de git que acompañaba al control hasta el checkout y hacía fallar 19 pruebas.

Analizador que verifica cada llamada FFI

Escribí un analizador que lee la cabecera C real de 7.308 líneas y verifica cada punto de llamada, cada constante de enumeración y que toda clase propietaria de un puntero sea final, después de que una cabecera provisional escrita a mano permitiera compilar y enlazar llamadas a tres funciones eliminadas que causaban fallos al ejecutarse.

Pruebas reducidas a 8,9 segundos

Pasé de seis pruebas que superaban un límite de sesenta segundos a 135 pruebas satisfactorias en 8,9 segundos al analizar el hilo principal y eliminar las dos llamadas en las que permanecía durante 3.989 de 4.017 muestras.

La mitad sin utilizar del núcleo, implementada

Implementé la mitad del núcleo de mensajería que la aplicación no utilizaba — transferencia de copias de seguridad, mensajes temporales, edición y reenvío de mensajes, invitaciones verificadas, proxies y política de cifrado —, ejecutando cada prueba contra la biblioteca real, sin simulaciones.

Conocer la licencia de todo lo que se publica

Audité la compatibilidad de licencias de 644 crates Rust en cada compilación y probé que el control se activa al cambiar la licencia de una crate y al modificar la revisión fijada del núcleo sin regenerar los datos.

Un criterio de historial para 4 repositorios

Apliqué un único criterio de historial a cuatro repositorios — commits convencionales, sin emojis ni líneas de atribución y con un hook de mensajes de commit —, junto con 48 documentos de instrucciones que regulan la forma de trabajar.

13 direcciones medidas sin rastreo

Medí trece direcciones únicamente a partir del registro de accesos del servidor web — sin cookies, rastreadores ni servicios de terceros —, enmascarando la red del cliente al escribir cada línea e incorporando el resultado a un archivo permanente de 21 recuentos diarios y 20 facetas mensuales.

Seis recolectores declaran lo que omitieron

Hice que seis recolectores de informes declararan la parte de sus entradas que nunca habían leído — archivos no abiertos, sondeos no instalados y líneas del diario no reconocidas — y ajusté los umbrales a 116 muestras medidas tras constatar que una recolección ciega y un servidor sano generan la misma página.

Centre el esfuerzo de ingeniería

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Sitúe el tema en su contexto

Cuéntenos cómo aparece «Automatización y CI/CD» en su propio sistema. Podemos ayudarle a elegir el servicio relevante y definir el trabajo en torno a él.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://platform.engineer.company/es/categories/automation/