Engineer

DevOps

44 réalisations

Reliez les parties de votre système

Les réalisations regroupées sous « DevOps » montrent une partie d'un système opérationnel. Nous pouvons vous aider à relier cette partie au reste de votre activité. Une supervision et des alertes qui signalent les incidents tant qu'ils restent mineurs, avec les chiffres qui montrent à quelle fréquence ils le restent.

Découvrir nos prestations Contactez-nous →

Trouvez un défi comparable

Suivez un exemple qui ressemble à votre situation. Son analyse décrit les décisions qui ont conduit au résultat, au-delà du nom de la technologie. Premier tableau de bord d'observabilité — première vue de l'entreprise sur ses propres systèmes

Premier tableau de bord d'observabilité Contactez-nous →

Premier tableau de bord d'observabilité

Développé et lancé le premier tableau de bord d'observabilité de l'entreprise, affichant en temps réel les performances du système et des visualisations de données sur le grand téléviseur des bureaux.

10 serveurs PostgreSQL et MS SQL sur Ubuntu

Conçu, déployé et maintenu 10 serveurs PostgreSQL et MS SQL sur des VPS Ubuntu Linux, en veillant à leurs performances et à leur fiabilité.

1 000 règles RBAC pour sécuriser Linux

Renforcé la sécurité des données en mettant en place 1 000 règles RBAC pour les développeurs, les instances applicatives, PostgreSQL, MS SQL et d'autres serveurs Linux afin de prévenir les accès non autorisés, avec une documentation et une automatisation Ansible.

Déploiement SaaS GIS automatisé : GitHub Actions

Automatisé le déploiement d'une application SaaS GIS, le traitement des données et le système de reporting avec GitHub Actions CI/CD, Python, Bash et SQL.

Livraison de 20 pipelines GIS automatisée

Automatisé la livraison de 20 pipelines de données GIS et processus ETL de données applicatives, en simplifiant l'automatisation de l'infrastructure et le reporting.

100 sauvegardes critiques automatisées : Barman

Automatisé 100 sauvegardes critiques avec Barman, Google Cloud, Bash et Python pour préserver l'intégrité des données dans les bases.

20 applications Docker sur Google Cloud et AWS

Déployé et maintenu 20 applications conteneurisées avec Docker, résolu des incidents avec Podman et Kubernetes et géré des applications R sur Google Cloud et AWS.

30 instances VPS Ubuntu Linux administrées

Géré 30 instances VPS Ubuntu Linux, mis en place des stratégies de reprise après sinistre et adapté les configurations réseau.

Risques réduits avec Grafana et Prometheus

Réduit les risques d'exploitation en mettant en place un tableau de bord de supervision avec Grafana et Prometheus pour améliorer la fiabilité du système.

Application GIS : revenus multipliés par 10

Dirigé le développement logiciel d'une application cartographique GIS, multipliant le chiffre d'affaires par 10 et faisant du produit un actif de données central.

SSL/TLS automatisé pour 100 applications Docker

Automatisé la création de certificats SSL/TLS pour 100 applications Docker afin de sécuriser les connexions sur des hôtes Ubuntu Linux.

CI/CD simplifiée : 4 000 heures gagnées

Simplifié les processus CI/CD et économisé 4 000 heures grâce à l'automatisation des pipelines de développement logiciel.

4 000 h gagnées avec GitHub et GitLab CI/CD

Simplifié les processus d'analyse de données et de développement logiciel et économisé 4 000 heures en introduisant des pratiques CI/CD avec GitHub, GitLab, Bash et Python.

Infrastructure simplifiée au-delà d'AWS

Refondu l'infrastructure d'un client international en supprimant les services inutilisés, en ajustant la capacité et en déplaçant les charges de travail hors d'AWS. La fondatrice fait état de coûts cloud divisés par 10, sans perte de productivité.

Contrat d'API généré à partir de la base

Généré le contrat d'API à partir de la base de données — OpenAPI, client TypeScript typé de 44 076 lignes, 61 gestionnaires de simulation et limites appliquées par l'interface — avec un contrôle à chaque étape qui échoue en cas de divergence.

Infrastructure Azure sous forme de code

Provisionné l'infrastructure Azure sous forme de code avec Bicep — Container Apps, PostgreSQL Flexible Server, Front Door/WAF et réseau — dans les environnements de développement, de préproduction et de production.

Pipelines CI/CD GitHub Actions

Conçu des pipelines GitHub Actions CI/CD avec une image frontend de production distroless et une promotion entre plusieurs environnements.

578 cibles d'automatisation go-task

Créé 578 cibles d'automatisation go-task couvrant les modes de développement natif, Docker et HTTPS, les contrôles de code, les tests, la base de données et le déploiement.

Déploiements Azure pris en charge de bout en bout

Pris en charge les déploiements de bout en bout sur Azure, en gérant les livraisons dans les environnements de développement, de préproduction et de production.

Sauvegardes et reprise sous forme de code

Configuré la conservation des sauvegardes sous forme d'Infrastructure as Code, puis évalué les possibilités de récupération et documenté la restauration, en explicitant les lacunes restantes avant qu'un incident ne les révèle.

Zéro avertissement dans six langages

Fixé un niveau d'exigence de zéro avertissement dans six langages — Go, TypeScript, SQL, Python, Shell et Markdown — appliqué par des hooks pre-commit.

Choix techniques fondateurs de la plateforme

Défini les choix fondateurs de la plateforme dans les semaines suivant l'ouverture du dépôt en novembre 2025 — architecture en couches, accès aux données centré sur la base et zéro avertissement — toujours en vigueur neuf mois plus tard.

Infrastructure centrale construite de zéro

Parmi les premières personnes recrutées, conçu et construit de zéro toute l'infrastructure centrale et les processus associés d'une startup SaaS d'énergie verte, posant les bases d'une croissance rapide.

Collaboration d'équipe automatisée

Automatisé la collaboration, la gestion des mots de passe, des tâches et du temps, et construit un système semi-automatique de présentation des projets pour accroître la productivité de l'équipe.

Suite de tests Go, SQL et navigateur

Conçu une suite de tests automatisés en couches — 981 tests Go, 543 spécifications frontend et navigateur, 494 tests de comportement SQL — avec tests par mutation, tests fondés sur les propriétés et contrôle obligatoire de l'accessibilité.

Moteur de 268 contrôles pour le dépôt

Conçu le moteur de garde-fous du dépôt — 268 contrôles de commit enregistrés, 277 règles de linting et 15 règles ESLint personnalisées — ainsi que 146 tests de ces garde-fous, pour que les exigences soient appliquées par la compilation et non par la seule revue.

Surveillance des erreurs et traçage internes

Conçu en interne la surveillance des erreurs et le traçage OpenTelemetry plutôt que de les acheter — assainissement des charges utiles, détection des pics et régressions, symbolication et signal de vie synthétique — accessibles dans 11 vues d'exploitation.

1 022 migrations contrôlées contre les écarts

Maintenu la cohérence du schéma au fil de 1 022 migrations avec un contrôle CI construisant la base de deux façons — installation neuve, puis installation suivie de toutes les migrations — et échouant en cas de différence.

Programme continu d'analyse de sécurité

Mis en place un programme continu de sécurité — analyse du code, DAST, contrôles des dépendances et des vulnérabilités, génération de SBOM, détection de secrets et actions épinglées par SHA — accompagné de 21 audits de sécurité écrits.

Une infrastructure définie dans un dépôt

Conçu l'infrastructure de l'entreprise avec 19 playbooks Ansible et 34 rôles totalisant 12 065 lignes de YAML, pour faire converger un hôte en service vers un état déclaré, chaque play étant idempotent.

Mode de vérification Ansible fiabilisé

Fiabilisé le mode de vérification sur toute la plateforme après avoir trouvé six sondes prenant des décisions sur une valeur que l'hôte n'avait jamais fournie : le module command d'Ansible annonce une réussite sous --check tout en sautant entièrement la commande.

Vérification des secrets en une seconde

Ajouté un play de vérification préalable exécutant en une seconde environ le même code que la convergence, avec les secrets locaux de l'opérateur, après qu'une exécution de production partiellement appliquée s'est arrêtée à la neuvième tâche, les paramètres de swap étant déjà écrits sur l'hôte en service.

Votre code sur votre propre serveur

Déployé la forge git de l'entreprise avec Soft Serve, privée par défaut, sans interface web et avec un port SSH lié à l'interface de bouclage derrière un hôte relais, puis fait de sa page d'accueil un artefact de compilation du site principal plutôt qu'une copie maintenue à la main.

Conteneurs exploités avec Podman et Quadlet

Déployé une couche de conteneurs Podman et Quadlet sous systemd plutôt que Docker, car la publication de ports sur le réseau bridge par défaut de Docker peut contourner les règles UFW, et confié les déploiements à un utilisateur non privilégié autorisé à une seule commande fixe plutôt qu'à root.

Contrôle avant commit avec 22 linters

Conçu un contrôle obligatoire avant commit composé de 22 linters tenant sur une ligne et de cinq nécessitant un paragraphe, sans niveau d'avertissement ni suppression locale autorisée, couvrant HTML, CSS, JavaScript, Python, YAML, Markdown, shell, liens, orthographe, secrets et typographie.

Des vérifications qui ne prennent plus une demi-heure

Réduit les vérifications qualité du site pilotées par navigateur de 1 636 à 615 secondes, en ordonnançant les contrôles du plus long au plus court dans un pool de quatre processus, après avoir mesuré un coût de 320 secondes en ordre alphabétique contre 224.

Contrôle qualité : 10 242 lignes vérifiées

Soumis 10 242 lignes de JavaScript des contrôles qualité à un formateur et à un linter, après avoir établi qu'il s'agissait du plus gros ensemble de code du dépôt et du seul que rien ne contrôlait, en corrigeant 13 constats sans en masquer aucun.

981 tests et 92 % de couverture

Vérifié le générateur avec 981 cas de test et un seuil de couverture des branches de 92 %, en traitant les avertissements comme des échecs, et vérifié l'idempotence en exécutant deux fois tout le pipeline depuis un fichier vide, sans changement autorisé au second passage.

Chaque règle du linter traitée comme une erreur

Activé comme erreurs toutes les règles du linter Python, corrigé 1 815 constats pour atteindre zéro et documenté la justification de chaque rare exemption, dont deux soutenues par un contrôle plutôt que par un commentaire.

Divergences entre hooks et contrôle qualité

Découvert que les hooks de commit et le contrôle qualité exécutaient des vérifications différentes alors qu'un document les disait identiques, en comparant les deux listes dans un test : les cinq contrôles seulement lancés à la main étaient ceux qui lisaient les textes du CV.

Hook CI de forge répété : 2 défauts détectés

Répété l'exécution du hook CI côté forge et trouvé deux défauts impossibles à atteindre par la seule lecture : un mécanisme de repli fournissant un argument impossible à résoudre à l'entrée du hook et une variable d'environnement git suivant le contrôle dans le checkout, faisant échouer 19 tests.

Suite de tests ramenée à 8,9 secondes

Fait passer la suite de six tests dépassant soixante secondes à 135 tests réussis en 8,9 secondes, en profilant le thread principal et en supprimant les deux appels où il restait bloqué pendant 3 989 des 4 017 échantillons.

Connaître la licence de tout ce que vous livrez

Audité à chaque compilation la compatibilité des licences de 644 crates Rust, puis vérifié le déclenchement du contrôle en modifiant la licence d'une crate et en déplaçant la révision épinglée du cœur sans régénération.

Une norme d'historique pour 4 dépôts

Appliqué à quatre dépôts une même norme d'historique — commits conventionnels, sans emojis ni mention d'attribution, imposés par un hook de message de commit — et 48 documents d'instructions régissant le travail.

Ciblez l'effort d'ingénierie

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Replacez le sujet dans votre contexte

Dites-nous comment « DevOps » se manifeste dans votre propre système. Nous pouvons vous aider à choisir la prestation pertinente et à définir les travaux correspondants.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/categories/devops/