Engineer

{ DevOps・CI/CD自動化 }

落ち着いて変更を届ける

変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

必要なことを整理している段階ですか？

うまくいっていないことや、つくりたいものをお聞かせください。何が必要か、おおよそいくらかかるか、私たちが適任かどうかをお伝えします。

時間単価は1,400 DKKから（VAT別）で、ご依頼期間が長いほど下がります。

お問い合わせ　料金を見る

このサービスに関する実績を抜粋してご紹介します。

サービスを成果につなげる

「DevOps・CI/CD自動化」を事業の成果につなげましょう。技術的な作業を、それが支えるべき利用者、運用、意思決定に結び付けるお手伝いをします。 変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

お問い合わせ

関連する実績を確かめる

このページの関連実績から、私たちの取り組み方をご確認ください。それぞれの成果の背景にある条件や、何を優先して何を見送ったかについてもお尋ねください。 GitHub ActionsでGIS SaaSのデプロイを自動化 — デプロイ、データ、報告を手作業から解放

GitHub ActionsでGIS SaaSのデプロイを自動化　お問い合わせ →

GitHub ActionsでGIS SaaSのデプロイを自動化

GitHub Actions CI/CD、Python、Bash、SQLで、GIS SaaSアプリケーションのデプロイ、データ処理、レポート作成を自動化した。

GISデータパイプライン20本の提供を自動化

20本のGISデータパイプラインとアプリケーションデータのETL処理の提供を自動化し、インフラ自動化とレポート作成を効率化した。

Barmanで重要データのバックアップ100件を自動化

Barman、Google Cloud、Bash、Pythonで重要データのバックアップ100件を自動化し、データベース間のデータ整合性を確保した。

Google Cloud・AWSでDockerアプリ20個を運用

Dockerでコンテナ化した20個のアプリケーションをデプロイ・保守。PodmanとKubernetesで問題を調査し、Google Cloud・AWS上のRアプリケーションも管理した。

GrafanaとPrometheusで運用リスクを低減

GrafanaとPrometheusの監視ダッシュボードを導入して運用リスクを低減し、システムの信頼性を高めた。

業務工程の刷新で8,000時間を削減

社内工程を見直し、ソフトウェア構成、システム、日程管理の効率化によって8,000時間を削減した。

Dockerアプリ100個のSSL/TLS証明書を自動化

Dockerアプリケーション100個のSSL/TLS証明書の作成を自動化し、Ubuntu Linuxホスト上での安全な接続を確保した。

CI/CDの効率化で4,000時間を削減

ソフトウェア開発パイプラインを自動化してCI/CD工程を効率化し、4,000時間を削減した。

GitHub・GitLab CI/CDで4,000時間を削減

GitHub、GitLab、Bash、PythonによるCI/CDを導入し、データ分析とソフトウェア開発の工程を効率化して4,000時間を削減した。

PythonとPL/pgSQLでデータ処理を自動化

Shellスクリプト、PL/pgSQL、Python、Transact-SQLでデータ処理を自動化し、生産性と効率を向上させた。

AWSの外へ移し、インフラを整理

海外顧客のインフラを見直し、未使用サービスを廃止して容量を調整し、ワークロードをAWSから移した。創業者の報告では、生産性を損なうことなくクラウド費用を10分の1に削減した。

データベースを起点にAPI契約を生成

データベースを起点に、OpenAPI、44,076行の型付きTypeScriptクライアント、61個のモックハンドラー、UIが適用する制限を生成し、各段階に不一致で失敗する検査を設けた。

GitHub Actions CI/CDパイプラインを構築

本番フロントエンド用のdistrolessイメージと複数環境間の昇格を備えた、GitHub ActionsのCI/CDパイプラインを構築した。

go-taskの自動化ターゲット578個を作成

ネイティブ、Docker、HTTPSの開発モード、静的検査、テスト、データベース、デプロイをカバーする、go-taskの自動化ターゲット578個を作成した。

Azureへのプラットフォーム導入を一貫して担当

プラットフォームのAzureへのデプロイを最初から最後まで担当し、開発・ステージング・本番環境のリリースを管理した。

6言語に警告ゼロの基準を設定

Go、TypeScript、SQL、Python、Shell、Markdownの6言語に警告ゼロの品質基準を設け、pre-commitフックで適用した。

基幹インフラ全体を一から構築

グリーンエネルギー分野のSaaSスタートアップで初期に採用された社員の1人として、基幹インフラ全体と運用工程を一から設計・構築し、急成長の基盤を整えた。

チームの協働を自動化

チームの協働、パスワード管理、タスク・時間管理を自動化し、プロジェクト実績を半自動で紹介する仕組みも構築して、チームの生産性を高めた。

Go・SQL・ブラウザーのテスト群を構築

981件のGoテスト、543件のフロントエンド・ブラウザーテスト、494件のSQL動作テストを持つ多層の自動テスト群を構築し、ミューテーションテスト、プロパティベーステスト、アクセシビリティの品質ゲートを加えた。

268検査を持つリポジトリの検査基盤を構築

登録済みコミット検査268件、静的検査ルール277件、独自ESLintルール15件からなるリポジトリの検査基盤と、検査自体を試す146件のテストを構築し、人のレビューだけに頼らずビルドで基準を守るようにした。

独自のエラー監視・トレースを構築

ペイロードの無害化、急増・再発の検知、シンボル解決、擬似ハートビートを備える独自のエラー監視とOpenTelemetryトレースを外部購入せず構築し、11画面の運用ビューから扱えるようにした。

1,022件のマイグレーションを差分検査で保護

新規インストールと、インストール後に全マイグレーションを適用する2通りでデータベースを構築し、不一致で失敗するCIゲートを設け、1,022件のマイグレーションにわたるスキーマの整合性を保った。

継続的なセキュリティ検査を運用

コード検査、DAST、依存関係・脆弱性検査、SBOM生成、秘密情報検査、SHA固定のActionsと、21件の文書によるセキュリティ監査を組み合わせた継続的なセキュリティ活動を確立した。

リポジトリで定義するインフラ

12,065行のYAMLにわたる19本のAnsible プレイブックと34個のロールで自社インフラを構築し、すべてのプレイを冪等にして、稼働中のホストを宣言した状態へ収束させた。

Ansibleチェックモードを実態に即した検証に

Ansibleのcommandモジュールが--checkでコマンドを省略しながら成功を返すため、ホストから得ていない値で判断していた6個の検査を修正し、プラットフォーム全体のチェックモードが実態を伝えるようにした。

秘密情報の1秒の事前検査を追加

本番実行が9番目のタスクで停止し、すでにスワップ設定が稼働ホストへ書き込まれた事態を受け、運用担当者のローカルな秘密情報に対し、収束処理と同じコードを約1秒で実行する事前検査プレイを追加した。

自社サーバーでコードを管理

Web管理画面を持たず、初期状態で非公開のSoft Serveによる自社gitホスティングを導入。SSHポートを踏み台ホストの背後のループバックに限定し、入口のページを手作業の複製ではなく主サイトのビルド成果物にした。

PodmanとQuadletでコンテナを運用

Dockerの既定のブリッジネットワークでのポート公開がUFWルールを迂回する場合があるため、systemd配下のPodman・Quadletでコンテナ基盤を構築。デプロイにはrootではなく、固定コマンド1つを使う非特権ユーザーを用いた。

22静的検査のコミットゲートを構築

HTML、CSS、JavaScript、Python、YAML、Markdown、Shell、リンク、綴り、秘密情報、組版を対象に、1行で説明できる検査22個と詳しい説明が必要な検査5個からなるコミットゲートを構築し、警告扱いや行内抑制を認めない方針にした。

30分も待たずに済む品質検査

最長の検査から順に、最大4並列のワーカープールへ割り当て、ブラウザーを使うサイト品質ゲートを1,636秒から615秒に短縮。名前順では224秒に対して320秒を要するという測定を踏まえた。

GeminiとGopherへサイトを複製

同じウェブサイトのソースからGemini版とGopher版を生成する仕組みを構築。この実績では、その時点で各形式777文書を記録し、形式の追加によって変更されたHTMLはゼロバイトだった。

更新日時が共通のサイトマップを修正

エクスポートが毎回すべてのファイルを書き直すことを確認し、git履歴から更新日を取得するよう変更。176件のURL中172件が同じ更新日時だったサイトマップを修正した。

10,242行の品質ゲート自体に静的検査を適用

リポジトリ最大のコード群でありながら検査対象外だった品質ゲートのJavaScript 10,242行に整形・静的検査を適用し、13件の指摘を抑制せずに修正した。

生成器に981テストと92%カバレッジを適用

警告を失敗扱いにし、分岐カバレッジ92%を下限として生成器に981件のテストを適用。空のファイルから全構築処理を2回実行し、2回目に何も変わらないことを要求して冪等性を検証した。

すべての静的検査ルールをエラーに設定

Python静的検査の全ルールをエラーとして有効にし、1,815件の指摘を0件まで解消。少数の例外には理由を書き、うち2件はコメントだけでなく検査によって根拠を守った。

検査自体のテストを作成

正常入力しか与えない検査は、何も読まずに正常と報告するようになり得ると確認し、検査自体のテストを作成。誤字を埋めて綴り検査の検出を確かめ、テスト内の固定値ではなくデータベースからID範囲を取得した。

フックと品質ゲートの不一致を発見

同じ検査を行うという文書の記述に反し、コミットフックと品質ゲートが異なる検査を行っていたことを、一覧を比較するテストで発見。手動でしか動かなかった5個は履歴書の文章を検査するものだった。

gitホスティングCIを試し2不具合を発見

gitホスティング側のCIフックを予行実行し、ファイルを読むだけでは到達できない2件の不具合を発見。解決不能な引数を入力へ渡すフォールバックと、検査先に引き継がれ19テストを失敗させるgit自身の環境変数を修正した。

全FFI呼び出しを検証する構文解析器を作成

手書きの仮Cヘッダーで、削除済みの3関数を呼んでもコンパイル・リンクが通り実行時に落ちた問題を受け、実際の7,308行のCヘッダーを読み、全呼び出し箇所、列挙定数、ポインター所有クラスのfinal指定を検証する構文解析器を作成した。

テスト群を8.9秒に短縮

メインスレッドを測定し、4,017サンプル中3,989サンプルを費やす2つの呼び出しを除去。6テストで60秒の制限を超えていたテスト群を、135テストが8.9秒で通る状態に改善した。

未使用だったコアの半分の機能を実装

バックアップ転送、消えるメッセージ、編集・再送、確認済み招待、プロキシ、暗号化方針など、アプリケーションが未使用だったメッセージングコアの半分の機能に対応し、モックなしで実ライブラリに対して全テストを実行した。

公開するすべての部品のライセンスを把握

毎回のビルドで644個のRust crateのライセンス互換性を監査し、1つのcrateのライセンスを書き換えた場合と、再生成せずコアの固定リビジョンを変えた場合に検査が働くことを実証した。

4リポジトリで共通の履歴基準を維持

4つのリポジトリに対し、Conventional Commits、絵文字禁止、帰属トレーラー禁止という共通の履歴基準をコミットメッセージフックで適用し、作業方法を定める48件の指示文書とともに運用した。

DevOps・CI/CD自動化をお探しですか？

リンク先の実績から、必要な経験があるかをご判断ください。つくりたいものをお聞かせいただければ、業務範囲、進め方、引き継ぎについてご相談いただけます。

お問い合わせ　すべての実績

発注前に業務範囲を定める

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る　お問い合わせ →

要件について話し合う

「DevOps・CI/CD自動化」に求めることを、納期や保守を担当するチームも含めてお聞かせください。実行可能な進め方をご相談いただけます。

お問い合わせ

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/services/devops-cicd/