GitHub による AI を活用した DevOps

効率性の向上、セキュリティの強化、
より迅速な価値提供

著作: GITHUB

目次

DevOps について

DevOps を効果的に導入することで、リリースサイクルの短縮、信頼性の向上、イノベーションの推進を実現できるため、組織がソフトウェアを提供する方法を大きく変えることができます。

DevOps を活用して急速に進化する市場で俊敏性を維持することで、真のチャンスを活用することができます。コラボレーション、継続的な改善、戦略的なテクノロジを導入する文化を確立することで、市場投入までの時間を短縮し、変化への適応力を高め、競合をリードすることができます。

DevOps は、多様な経験、技術的スキル、文化的な視点によって形作られます。この多様性が、幅広い解釈と進化する実践をもたらし、DevOps をダイナミックで学際的な分野にしています。DevOps チームは、ソフトウェア開発ライフサイクル (SDLC) の一部を担うチームの主要メンバーで構成される、機能横断的なチームです。

この eBook では、強力な DevOps チームと実践を構築することの価値、そしてルーチンタスクの自動化、コードの保護、そして最適なエンド ツー エンドのライフサイクル管理を実現する AI の活用方法について掘り下げます。

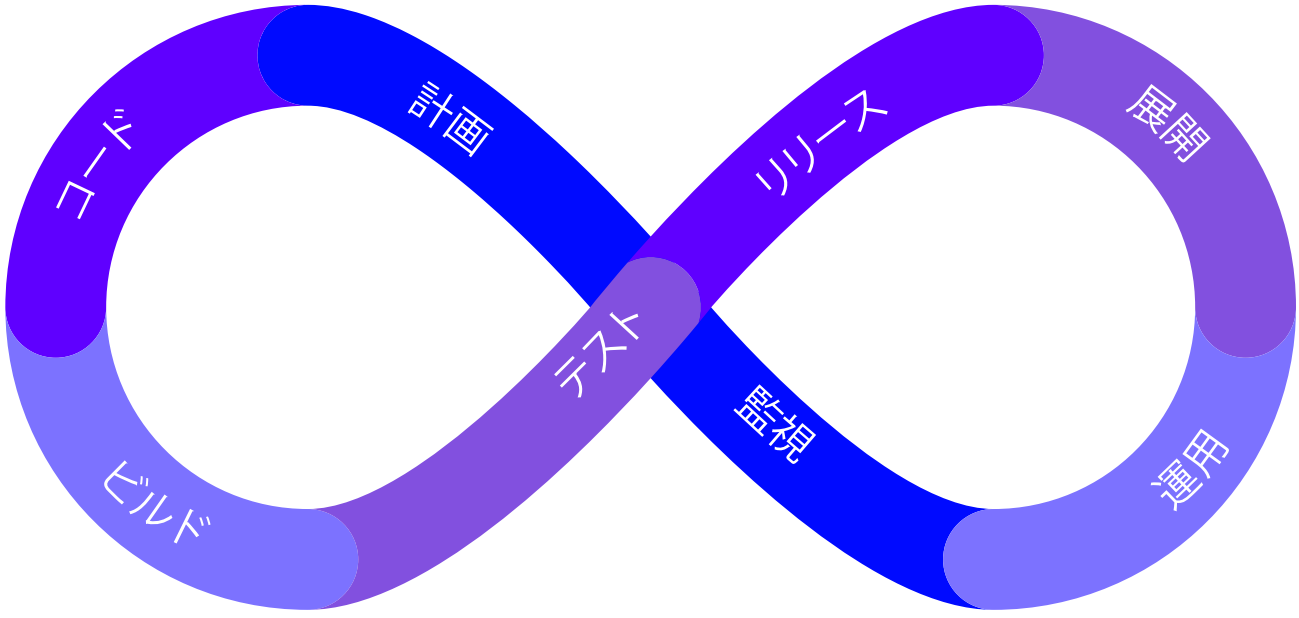


図 1: DevOps の継続的なライフサイクル

DevOps の定義

DevOps コミュニティで高く信頼されている Donovan Brown 氏が、DevOps の定義を提示しており、DevOps の実践者たちに広く認知されています。

DevOps とは、エンド ユーザーに継続的に価値を提供する、人材、プロセス、製品の連携を表すものです。”

Donovan Brown

パートナー プログラム マネージャー // マイクロソフト[1]

多くの技術環境では、チームは各自の技術スキルセットによってサイロ化されており、それぞれが独自の指標、KPI、成果物に焦点を当てています。この断片化は、開発の遅延を引き起こします。非効率性をもたらし、優先順位の競合につながることで、最終的には進歩を妨げます。

これらの課題を克服するためには、コラボレーションの促進、建設的なフィードバックの奨励、ワークフローの自動化、継続的な改善を受け入れる必要があります。これによって、より迅速なソフトウェアの提供、効率性の向上、意思決定の改善、コスト削減、より強力な競争優位性の強化を実現できます。

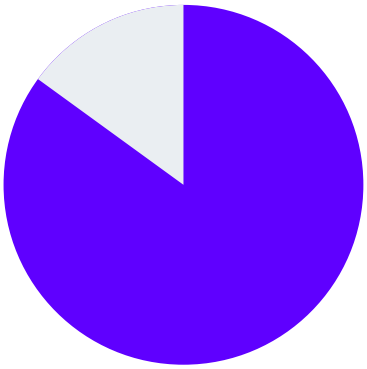
それではチームがどのようにすれば、新しい DevOps の実践を効果的に導入していけるでしょうか? 手作業による展開プロセス、長いフィードバック サイクル、非効率的なテスト自動化、リリース パイプラインへの手動介入による遅延など、最も深刻な問題点への対処から始めることができます。

1: https://www.donovanbrown.com/post/what-is-devops

摩擦点を排除することは、非常に大変な作業のように感じられますが、近年急速に発展している AI は、開発者の作業のスピードと品質を高める新たな機会を生み出しています。GitHub による調査の結果、GitHub Copilot Chatを有効にすると、この機能の使用経験がなくても、作成されたコードやレビューされたコードの品質が全体的に向上することがわかりました。

開発者の 85% は、GitHub Copilot と GitHub Copilot Chat を使ってコードを作成すると、コードの品質に自信を持てたと感じています

85%



GitHub Copilot Chat を使用しない場合と比較して、コード レビューをすぐに始められるようになり、完了までの時間が 15% 短縮されました

15%

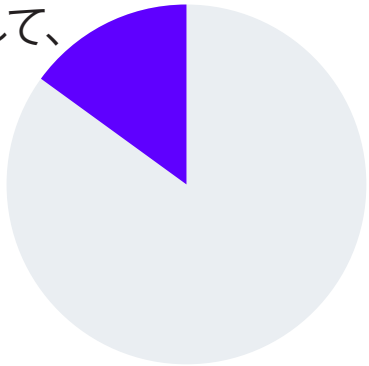


図 2: AI の導入が開発者に与える影響[2]

2: https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-code-quality/

DevOps + 生成 AI: AI を使用して効率性を高める

責任を共有する文化を推進することで、DevOps はコラボレーションを促進し、サイロ化を解消します。AI によってこの取り組みをさらに前進できます。反復的な作業を自動化し、ワークフローを合理化し、より迅速なフィードバック サイクルを実現することで、価値の高い作業に集中できるようになります。

ソフトウェア開発における主な課題は、非効率性と不正確性です。AI は、リソース管理を最適化し、一貫性のある正確な成果を提供することで、これらの問題解決を支援します。AI 主導による効率化では、アプリケーションのパフォーマンスとインフラの最適化を向上させるだけでなく、セキュリティを強化し、コスト削減も実現できます。

生産性の高いチームは、生産性を妨げ、開発サイクルを延長させる反復的な作業を特定して、自動化することができます。最終的な目標は、組織の成長を促進し、市場投入までの時間を短縮し、開発者の生産性と満足度を高めると同時に、顧客とエンドユーザーにとって最も重要なものを提供することです。

雑務の自動化

開発者は、反復的な日常業務を処理することが一般的です。これらは一般的に「時間泥棒」と呼ばれ、手動でのシステム チェック、新しいコード環境の設定、バグの特定と修正などが含まれます。これらの作業は、開発者が本来担うべき新機能の開発という業務から時間を奪ってしまいます。

DevOps では、チームの連携と自動化が均等に行われます。包括的な目標は、SDLC から負担や障壁を取り除き、開発者から手作業の単調な作業を削減できるようにサポートすることです。AI を活用して、これらの問題を解決する方法を見てみましょう。

図 3: 開発者の効率を低下させるアクティビティ

GitHub で開発ライフサイクルを合理化する

DevOps、AI、GitHub の機能を組み合わせ、チームがどのようにしてエンド ツー エンドの価値を提供できるかを見てみましょう。GitHub はオープンソース ソフトウェアのホームとして広く認知されていますが、GitHub Enterpriseのソリューションを通じてエンタープライズ レベルの機能も提供しています。

GitHub Enterprise は、バージョン管理、課題追跡、コード レビューなどの統合プラットフォームを提供することで、DevOps のライフサイクルを合理化します。これにより、ツールチェーンの無秩序な増大と非効率性を最小限に抑えられると同時に、チームが作業する攻撃対象領域を削減できるため、セキュリティ リスクを軽減できます。

主要な AI 開発ツールである GitHub Copilot を利用することで、繰り返されるタスクに費やす時間を減らし、エラーを軽減して開発サイクルを加速できます。これにより、より迅速な開発と市場投入までの時間短縮を実現できます。

GitHub に搭載された自動化と CI/CD ワークフローは、コード レビュー、テスト、展開の簡素化にも役立ちます。これにより、手作業タスクの削減、承認時間の短縮、開発の加速を実現できます。これらのツールによってシームレスなコラボレーションとサイロ化の解消を実現できるため、チームがプロジェクトのあらゆる側面で、計画から納品までを効率的に管理できるようになります。

「より懸命に」ではなく「より賢明に」

自動化は DevOps の核心部です。これによって時間の無駄をなくし、より迅速な価値提供に集中できるようになります。自動化という言葉には、SDLC のさまざまな項目が幅広く含まれています。自動化には、CI/CD を構成してコードの変更を実稼働環境にシームレスに統合することも含まれています。これには、コードとしてのインフラ (IaC)、テスト、監視とアラート、セキュリティの自動化も含まれます。

ほとんどの DevOps ツールでは CI/CD 機能を利用できますが、GitHub はさらに一歩進んで、クラウド、オンプレミス、その他の環境を問わず、あらゆる環境にエンタープライズグレードのソフトウェアを提供します。GitHub Actions を使用すると、CI/CD パイプラインをホストするだけでなく、ワークフロー内の事実上すべてを自動化できます。

GitHub プラットフォームとのシームレスな統合により、余分なツールが不要になり、ワークフローが合理化されるため、生産性が向上します。GitHub Actions がワークフローをどのように変革できるかを次に示します。

- **CI/CD** の高速化: ビルド、テスト、展開のパイプラインを自動化し、リリースを迅速化します。
- コード品質の向上: コード フォーマットの標準化を徹底し、セキュリティ上の問題を早い段階で発見します。

- コラボレーションの強化: 開発のプロセスに関する通知と通信を自動化します。
- コンプライアンスの簡素化: リポジトリを組織の標準に準拠させることができます。
- 効率性の向上: 反復性のあるタスクを自動化して、開発者の時間を解放します。

GitHub Copilot は、コードの提案や、より良いワークフローを作成するために使用するアクションを提案できます。また、組織に合わせたコーディングのベスト プラクティスを提案するため、チームにすぐに導入して、ガバナンスと規約の強化に役立てることができます。GitHub Copilot は、さまざまなプログラミング言語とも連携し、タスクを簡単に自動化するアクションやワークフローの構築にも使用できます。

GitHub Copilot の詳細情報については以下をご参照ください。

- GitHub Copilot で IDE にコードの提案を表示させる
- GitHub Copilot を IDE で使用する: ヒント、コツ、ベスト プラクティス
- GitHub Copilot の意外な 10 の使用方法

反復性のあるタスク

日常的な作業の自動化に重点を置いて、GitHub Copilot などのツールを使用してワークフローを合理化します。たとえば、Copilot はソフトウェア開発において、時間がかかり、不可欠な部分である単体テストの作成を支援できます。開発者は、正確な指示を作成することで、Copilot に基本的なシナリオとより複雑なケースの両方を網羅した包括的なテスト スイートを作成させることができます。これにより、コードの高い品質を維持しながら手作業を減らすことができます。

Copilot が提供する結果は、信頼しつつも検証が不可欠となります。これは AI を搭載した生成ツール全般に共通することです。チームで活用する際には、単純な作業から複雑な作業まで、あらゆる作業を Copilot に任せることができますが、コードを展開する前に、出力結果を徹底的なテストで常に検証していくことが重要です。これにより、信頼性を確保するだけでなく、ワークフローの速度を低下させる可能性のあるエラーも予防できます。

Copilot を使い続けるうちに、プロンプトを改良していくことで、機能を最大限に活用できるようになります。よりスマートな自動化が可能になり、反復性のあるタスクをさらに最小限に抑えることができます。

GitHub Copilot を使用した単体テストの作成の詳細については、以下を参照してください。

- GitHub Copilot のツールを使用して単体テストを作成する
- GitHub Copilot でテストを作成する

プロンプト エンジニアリングとコンテキスト

GitHub Copilot を DevOps プラクティスに統合することで、チームの働き方を大きく変えることができます。Copilot に正確で状況に応じた指示を出すことで、チームの効率性を上のレベルに引き上げ、プロセスを合理化することができます。

これらのメリットは、以下のような組織に測定可能な成果をもたらします。

- 効率性の向上: 反復性のあるタスクを自動化し、手動による介入を最小限に抑え、実用的なインサイトをもたらすことで、より迅速かつスマートな意思決定を可能にします。
- コスト削減: 反復的でエラーが発生しやすいプロセスに AI を統合することで、ワークフローを合理化し、エラーと開発コストを削減します。
- 好結果につなげる: Copilot を活用することで、戦略目標の達成、カスタマー エクスペリエンスの向上、市場での競争優位性を維持できます。

正確で詳細な指示の書き方を身につけることで、チームは Copilot の提案の関連性と精度を大幅に向上させることができます。新しいツールを導入する際には、チームが Copilot のメリットを最大限に活用できるよう、適切な導入とトレーニングが不可欠です。

チーム内で効果的なプロンプト エンジニアリングの文化を育む方法は次のとおりです。

- **社内コミュニティの構築**: インサイトの共有、イベントへの参加や主催、学習機会を創出するチャット チャネルを設定して、チームが学ぶためのスペースを作ります。
- **意外な瞬間を共有する**: Copilot などのツールを使用して、他のユーザーの取り組みのガイドとなるドキュメントを作成します。
- **気になったヒントやテクニックを共有する**: ナレッジの共有セッションを開催し、社内コミュニケーション (ニュースレター、Teams、Slack など) を使用してインサイトを共有します。

効果的なプロンプトを使いこなすことで、AI をチームの目標と合わせる際に役立ちます。そしてより良い意思決定、より信頼性の高いアウトプット、より高いパフォーマンスにつなげることができます。これらのプロンプト エンジニアリングの手法を導入することで、コスト削減だけでなく、より迅速な納品、より充実した製品ラインナップ、より優れたカスタマー エクスペリエンスを実現することができます。

DevOps + セキュリティ: コードを徹底的に保護する

統一された戦略で SDLC を管理することは、合理化されたツールセットをサポートしている場合には、はるかに効果的となります。ツールの乱立は、多くの DevOps 分野に共通する課題ですが、その影響を最も強く受けることが多い領域はアプリケーションのセキュリティです。チームではギャップを埋めるために新しいツールを追加することが多くありますが、このアプローチでは、人やプロセスに関する根本的な問題を見落としがちです。その結果、セキュリティの環境が、単一アプリケーションのスキャナーから複雑なエンタープライズ リスク プラットフォームまで、あらゆるもので乱雑になってしまう可能性があります。

ツールセットを簡素化することで、開発者は集中力を維持し、コンテキスト切り替えを減らし、コーディングのフローを維持することができます。依存関係の管理や脆弱性に関するアラートから、機密情報を保護する予防措置に至るまで、あらゆる段階でセキュリティが統合されたプラットフォームによって、組織のソフトウェア セキュリティの安定性を高めることができます。さらに、拡張性も重要です。プラットフォームに搭載された機能と併せて、既存のツールを活用することができます。

コードのあらゆる行を保護する

ソフトウェアの開発といえば、Python、C# 、Java、Rust などの言語が思い浮かぶことでしょう。しかし、コードにはさまざまな形態があり、データ サイエンティスト、セキュリティ アナリスト、ビジネス インテリジェンス アナリストなど、さまざまな分野の専門家も、それぞれ独自の方法でコーディングに取り組んでいます。結果として、セキュリティ脆弱性に対する潜在的なリスクも増加します。あるいは、気づかないうちにリスクが高まっている場合もあります。役割や肩書きに関係なく、すべての開発者に包括的な基準と手法を提供することで、開発者はサイクルのすべての段階にセキュリティを組み込むことができます。

静的分析とシークレット スキャン

ビルド時の統合に関しては、アプリケーション セキュリティ テスト (AST) ツールを使用することが一般的になっています。最小限の侵襲性備えた手法では、ソース コードをそのままスキャンし、複雑な箇所、潜在的な悪用、標準への準拠の有無を調査します。すべてのコミットとプッシュに対してソフトウェア構成分析 (SCA) を使用することで、開発者は目の前のタスクに集中すると同時に、プル リクエストやコード レビューの生産性と意義を高めるメカニズムを提供できます。

シークレット スキャンは、ソース管理に潜在的に危険なシークレットやキーをコミットしてしまうことを防ぐ秘密兵器です。シークレット スキャンを構成することで、AWS、Azure、GCP など、120 社を超えるソフトウェアとプラットフォーム ベンダーのリストから引き出すことができます。これにより、特定のソフトウェア アプリケーションやプラットフォームと一致する特定のシークレットを識別できます。GitHub の UI から直接、シークレットやキーがアクティブであるかどうかをテストすることもでき、修正を簡単に行うことができます。

CodeQL による高度なコード分析

CodeQL はコードを分析して脆弱性、バグ、その他の品質問題を特定することができる、GitHub の強力なユーティリティです。コンパイルまたは解釈を通じてコードベースからデータベースを構築し、脆弱なパターンを検索するクエリ言語を使用します。また、CodeQL では、特定のケースやビジネスに関連する独自の使用用途に合わせてカスタマイズした、カスタム バリアント データベースを作成することもできます。このような柔軟性を備えているため、組織内の他のアプリケーションのスキャン時に使用できる再利用可能な脆弱性データベースの開発ができます。

CodeQL は、その強力な機能に加え、サポートされている言語のスキャン結果と脆弱性結果を迅速に提供するため、開発者は品質を損なうことなく効率的に問題に対処できます。このパワーとスピードの組み合わせにより、CodeQL はさまざまなプロジェクトにおけるコードの整合性とセキュリティを維持する上で貴重な資産となります。また、組織のリーダーに対して、回復力を向上させ、セキュアなソフトウェア開発手法を導入する拡張可能なアプローチを提供します。

28
分

脆弱性の検知から
修復の完了まで[3]

2.4 倍
の精度

低い誤検出で 、
漏洩した機密を検出[4]

90%
のカバレッジ

Copilot Autofix は、サポートされているすべての言語において、アラートのほぼ 90% のタイプにコードの提案を提供[5]

3: 全体として、開発者が Copilot Autofix を使用して PR タイム アラートの修正を自動的にコミットするまでの時間は中央値で 28 分でした。同じアラートを手動で解決した場合には 1.5 時間かかりました (3 分の 1 に短縮)。SQL インジェクションの脆弱性の場合は、3.7 時間に対して 18 分でした (12 分の 1 に短縮)。GitHub Advanced Security が有効になっているリポジトリのプル リクエスト (PR) において、CodeQL によって検出された新しいコード スキャン アラートに基づく。これらは例です。実際の結果は異なる場合があります。

4: A Comparative Study of Software Secrets Reporting by Secret Detection Tools、Setu Kumar Basak 他、ノースカロライナ州立大学、2023年

5: https://github.com/enterprise/advanced-security

依存関係グラフの謎を解き明かす

最新のアプリケーションでは、直接参照されるパッケージが数十個存在し、さらにそれらのパッケージに依存するパッケージが数十個存在する場合もあります。企業はさまざまなレベルの依存関係を持つ数百ものリポジトリ管理に直面しているため、この課題はさらに深刻化しています。これによって、組織全体でどの依存関係が使用されているかを把握することが難しくなるため、セキュリティ対策は困難な作業となります。リポジトリの依存関係、脆弱性、OSS ライセンスの種類を追跡する依存関係管理戦略を採用することで、リスクを軽減し、実稼働環境に到達する前に問題を検出することができます。

GitHub Enterprise は、ユーザーと管理者に依存関係グラフに関するインサイトを即座に提供し、Dependabot からの使用アラートとともに、潜在的なセキュリティ リスクとなる古いライブラリを警告します。

リポジトリの依存関係グラフの構成要素は次のとおりです。

- **依存関係**: リポジトリで識別された依存関係の完全なリスト
- **依存先**: リポジトリに依存するプロジェクトまたはリポジトリ
- **Dependabot**: 依存関係の更新バージョンに関する Dependabot による調査結果

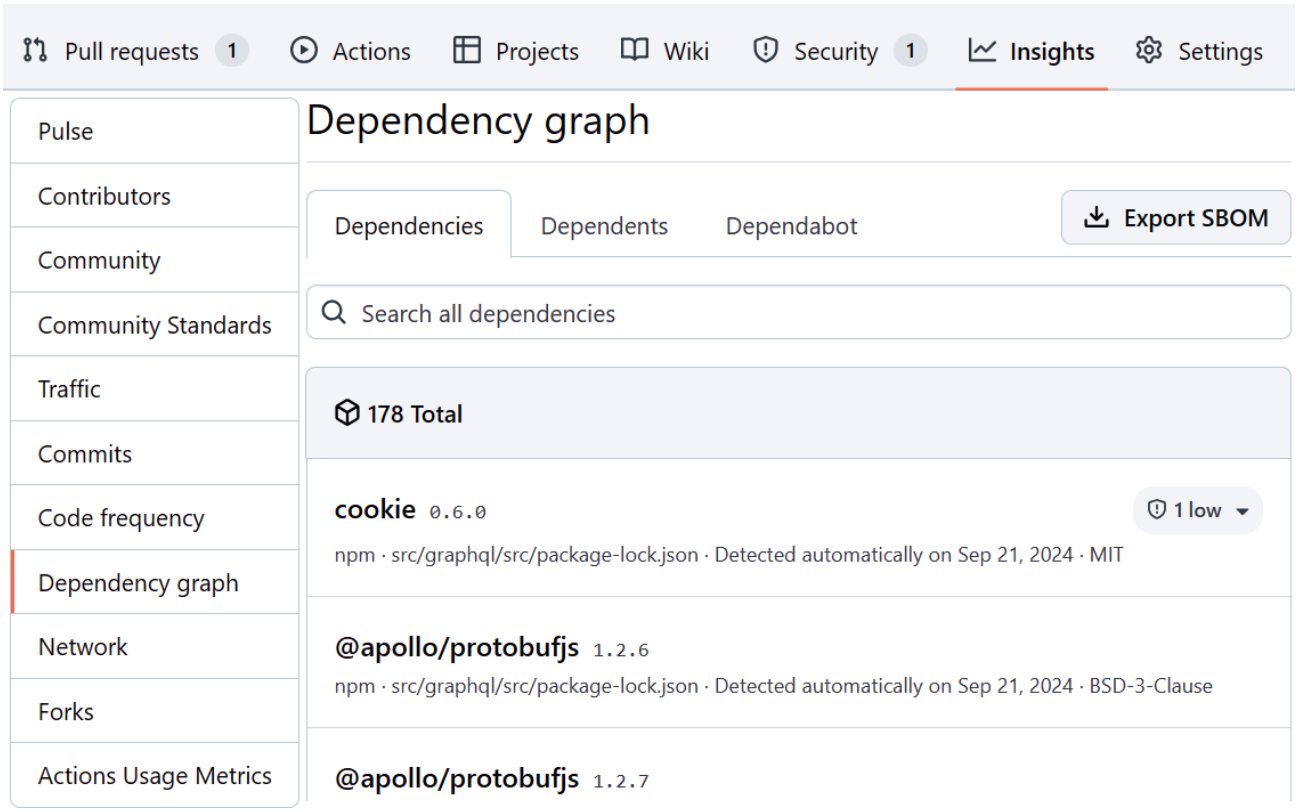


図 4: 依存関係グラフ

リポジトリ レベルの脆弱性については、ナビゲーション バーのセキュリティ タブに、ご利用のコードベースに関連する依存関係に関連付けられている可能性がある、特定された脆弱性の結果が表示されます。Dependabot のビューには、特定された脆弱性に関連するアラートが一覧表示され、パブリック リポジトリの特定のアラートを自動的に優先順位付けする際に参考にできるルールセットを表示できます。

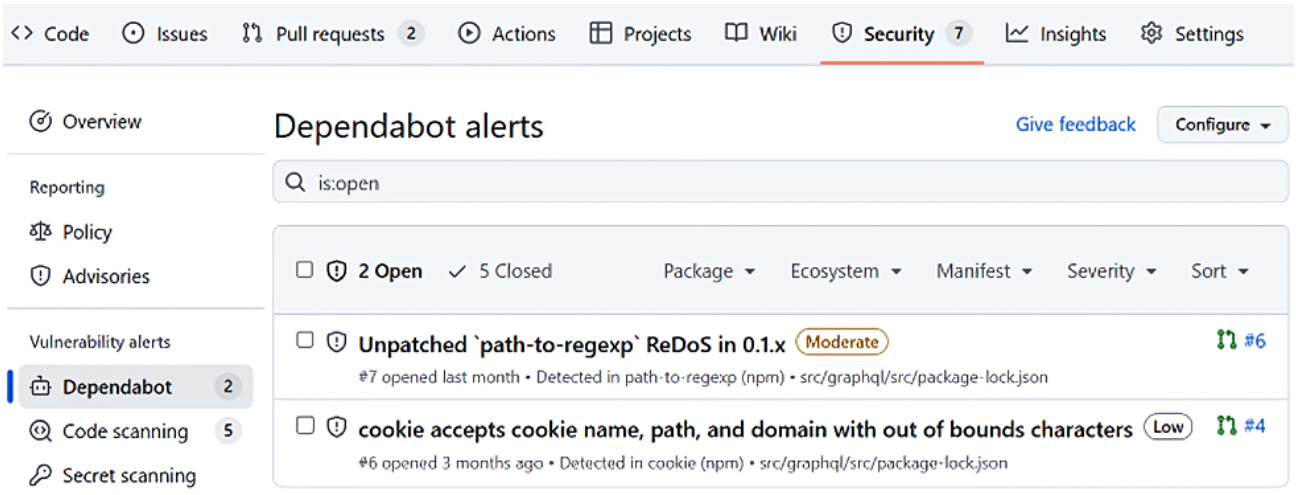


図 5: 脆弱性アラート

GitHub エンタープライズと組織のビュー

GitHub Enterprise を使用すると、組織と企業内のすべてのリポジトリにわたる依存関係、脆弱性、OSS ライセンスを閲覧、管理することができます。依存関係グラフでは、登録済みのすべてのリポジトリにわたる依存関係の包括的なビューを確認できます。

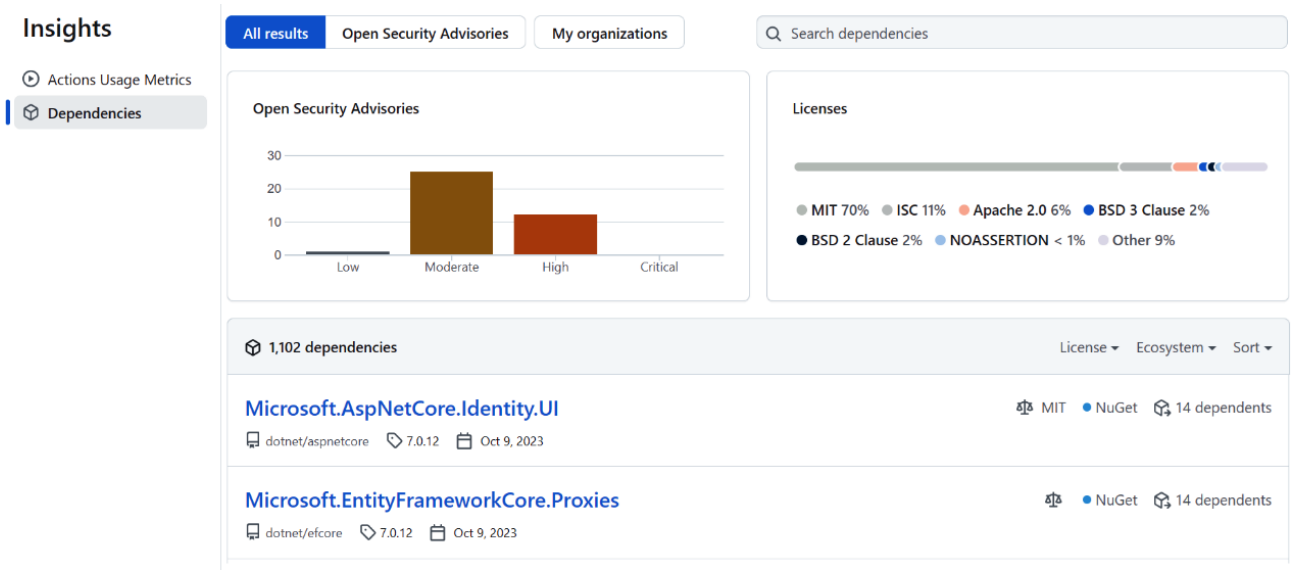


図 6: 依存関係、脆弱性、OSS ライセンスの表示と管理

この視認性に優れたダッシュボードは、特定されたセキュリティ アドバイザリだけでなく、企業全体で使用されている依存関係に関連するライセンスの分布状況についても、優れた概要情報を提供します。OSS ライセンスの使用は、特にプロプライエタリ コードを管理している場合は特にリスクが高い場合があります。GPL や LGPL など、より制限の厳しいオープンソース ライセンスでは、ソース コードが強制公開の対象となる脆弱性が潜在的に残っている可能性があります。オープンソース コンポーネントでは、コンプライアンス違反の可能性を判断するための統一された方法を見つけ、それらのライセンスで取り込まれているパッケージの代替品を見つける必要があります。

セキュリティ態勢を強化する

多くのエンタープライズ グレードのソース制御管理システムでは、ポリシー、コミット前のフック、プラットフォーム固有の機能を使用してコードを保護するオプションが用意されています。次の手段を使用して、包括的なセキュリティのスタンスを計画できます。

- **予防策**: GitHub では、特定のブランチにおける望ましくない変更を防止し、動作を強制するために、さまざまなタイプのルールセットを設定し、使用することができます。次に例を示します。

  › 変更をマージする前にプル リクエストを必要とするルール

  › 特定のブランチが直接変更をプッシュしないように保護するルール

  追加のクライアント側チェックは、コミット前フックを使用することで実行できます。Git はソース制御管理システムとして、コミット前フックをサポートしており、コミット メッセージの書式設定や、変更をコミットする前に書式設定や検証ルーチンを実行するなど、さまざまなタスクを実行できます。これらのフックでは、高度なユーティリティを利用してローカル レベルでコードの一貫性と品質を確保できます。

- **保護対策**: GitHub では、プル リクエストや CI ビルド時に構成できるチェック機能の使用など、保護対策の設定も可能です。これには以下が含まれます。

  › 依存関係のチェック

  › テストのチェック

  › コード品質のチェック

  › 品質ゲート

  › 手動介入/人間による承認ゲート

GitHub Enterprise を活用することで、ソフトウェア開発チームが、古い依存関係やチェックインされた機密情報から既知の言語の脆弱性まで、脆弱性を迅速に特定し、対応できるようになります。依存関係グラフを表示する機能が追加されたことで、チーム リーダーや管理者は、セキュリティ アドバイザリに関して常に先手を打つために必要なツールを備えることができます。使用中のライセンス タイプの可視性を提供することで、包括的なセキュリティ優先のリスク管理プラットフォームとして活用できます。

GitHub Enterprise で DevOps のパイプラインを強化する

DevOps の概念は、すでにテクノロジ業界では広く知られていると言っても過言ではありません。しかし、アプリケーションを展開する新しいツールや方法論が次々と登場する中、成長を続ける組織が結果を効果的に管理し、測定することは困難です。

回復力、拡張性、コスト効率に優れたアプリケーションに対する市場の需要に応えることは、難しい場合があります。クラウドベースのリソースを活用することで、市場投入までの時間を短縮し、開発者向けの内部ループを高速化し、コストを意識した管理のもとで、規模に応じたテストと展開を行うことができます。

クラウド ネイティブ アプリケーションの有効化

シフトレフトのパラダイムがセキュリティ、テスト、フィードバックを開発のインナー ループに近づけたように、クラウド用アプリケーションの開発についても同じことが言えます。クラウド中心の開発手法を採用することで、開発者は従来のアプローチと最新のクラウド ソリューションの間のギャップを埋めることができます。この移行により、チームはクラウド ファーストのアプリケーションを単に作成するだけでなく、真のクラウドネイティブなアプリケーションを構築できるようになります。

クラウドで開発してクラウドに展開する

シームレスな開発を促進する IDE は、現在では標準となっています。しかし、その環境内での移植性という概念は比較的新しく、特にクラウドベースの IDE の最近の進歩を考慮すると、その傾向が顕著であると言えます。GitHub Codespaces とそれを支える DevContainers テクノロジの登場により、開発者は移植可能なオンライン環境でコードを開発できるようになりました。この設定により、構成ファイルを利用できるようになるため、開発環境を特定のチームの要件に合わせてカスタマイズできるようになります。

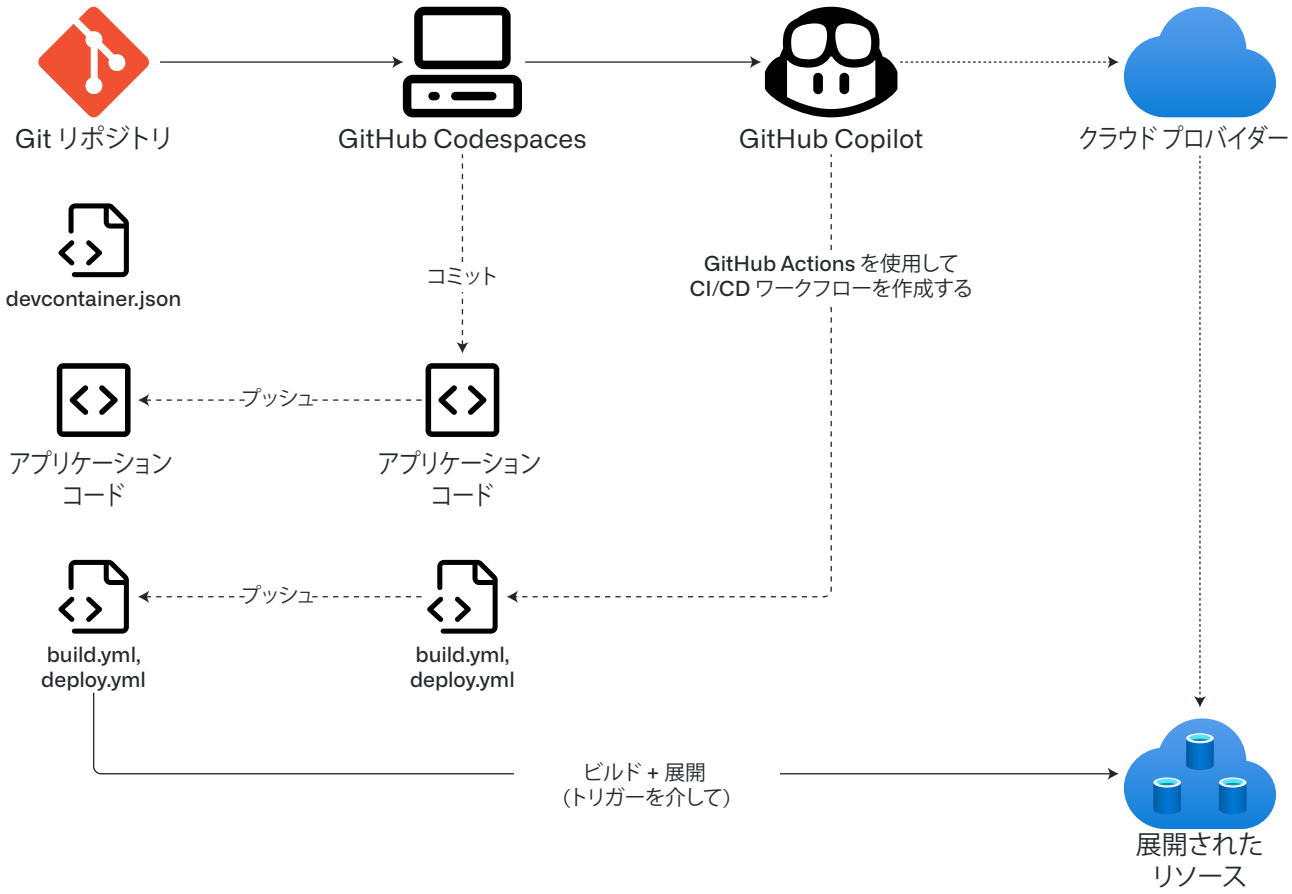


図 7: ポータブル オンライン環境

再利用性と移植可能性の組み合わせは、組織に大きな利点をもたらします。チームは、構成と環境の仕様を一元管理できるようになります。これによって、経験の浅い開発者でもベテランの開発者でも、同じ設定で作業できるようになります。これらの一元管理された構成により、チーム メンバーはそれらの構成に貢献することができます。ニーズの進化に合わせて環境を更新し、すべての開発者にとって安定した状態を維持することができます。

大規模なワークフローの管理

生産性の指標を実際に左右するのは、開発者のワークフローと市場投入までの時間です。しかし、これを大規模に管理することは、特にさまざまな開発チームがワークフローや展開をさまざまなクラウド、クラウド サービス、あるいはオンプレミス環境で使用している場合には、困難な場合があります。GitHub Enterprise が大規模なワークフロー管理の負担を軽減する方法をいくつかご紹介します。

- 再利用可能なアクションとワークフローで簡素化する
- アクション ポリシーを使用したガバナンスを採用する
- 検証済みパブリッシャーが公開しているアクションを使用する
- ブランチ ポリシーとルール セットを使用して一貫性を確保し、メインライン コードを保護する
- エンタープライズ レベルと組織レベルで適切な構成を行う

エンド ツー エンドのソフトウェア ライフサイクル管理

計画された作業と進行中の作業の両方を管理することは、アジャイルなソフトウェア開発の重要な基盤です。GitHub Enterprise は、軽量なプロジェクト管理構造を提供します。ここでは、ユーザーがプロジェクトを作成し、1 つ以上のチームとリポジトリをそのプロジェクトに関連付けでき、リンクされたリポジトリで開かれた問題を使用して、プロジェクト内の作業項目を全体的に追跡することができます。ラベルを使用して、さまざまな種類の問題を区別できます。

たとえば、課題に使用できるデフォルトのラベルには、機能拡張、バグ、機能などがあります。その課題に関連するタスク リストがある場合、Markdown を使用してそのタスク リストをチェックリストとして定義し、課題の本文に含めることができます。これにより、そのチェックリストに基づいて完了の追跡ができるため、定義されている場合は、プロジェクトのマイルストーンと整合させることができます。

フィードバック ループの管理

開発者が特定の機能に関するフィードバックを早いタイミングで受け取れば、変更を検証することに比べて、潜在的な問題の修正と更新プログラムのリリースを容易に実行できます。これは周知の事実です。インスタント メッセージ、メール、チケットや問題に関するコメント、あるいは電話など、あらゆる組織にはそれぞれ推奨するコミュニケーション方法があります。GitHub Enterprise の追加機能の 1 つに「ディスカッション」があります。ここでは、開発者とユーザーがフォーラムベースの環境でやりとりを行い、変更内容や機能に関するあらゆる問題、または新しい機能に関する提案を作業項目に変換できます。

ディスカッションに関する機能セットは、オープンソース プロジェクトで長い間人気を博してきました。すでに企業レベルのコミュニケーション ツールが導入されている場合、ディスカッションを使用するメリットがわかりにくいと感じる組織もあるかもしれません。組織が成熟していくにつれて、特定のソフトウェアの機能や性能に関連するコミュニケーションを分離し、それらを特定のリポジトリに関連付けられたディスカッションを通じて伝達することができるようになります。これによって、開発者、製品オーナー、エンド ユーザーが、関心のある機能に特化した環境で緊密に連携する機会を得られるかもしれません。

成果物のライフサイクル

ソフトウェア開発のライフサイクルすべてにおいて、その中心となるものに成果物の管理が挙げられます。実行可能ファイル、バイナリ、動的なリンク ライブラリ、静的な Web コード、あるいは Docker コンテナ イメージや Helm チャートであっても、展開のすべての成果物をカタログ化し、取得できる一元的な場所を持つことは不可欠です。GitHub Packages によって、開発者が組織内や企業内で配布する標準化されたパッケージ形式を保存できるようになります。

GitHub Packages は以下をサポートしています。

- Maven
- Gradle
- npm
- Ruby
- .NET
- Docker イメージ

これらのカテゴリに該当しない成果物がある場合でも、リポジトリのリリース機能を使用して保存することができます。これにより、必要に応じて必要なバイナリやその他のファイルを添付できます。

品質管理

テストについては、継続的インテグレーションのビルド中における単体テストや機能テスト、品質保証アナリストが Web アプリケーション内の機能を検証するテスト シナリオの実行など、ソフトウェア開発には欠かせない要素です。GitHub Actions を利用すると、さまざまな種類のテストをパイプラインに統合できるため、品質評価の実施を徹底できます。さらに、GitHub Copilot は、単体テストの作成方法に関する最適な提案を提供できるため、単体テストやその他のテストを作成する負担を開発者から取り除きます。これによって、開発者は目の前のビジネス上の問題にさらに集中できるようになります。

さまざまなテスト ユーティリティを簡単に統合できるため、開発のライフサイクル全体にわたって品質を確実に評価できるようになります。前述の通り、GitHub Actions のワークフロー内でチェックを使用して、特定のシナリオを検証できます。これには、リクエストをマージする前に、一連のテストをすべて正常に実行できることが含まれます。展開の段階に応じて、統合テスト、負荷テスト、ストレス テスト、さらにはカオス テストなどのチェックを指定することもできます。これによって、展開のパイプラインを通過するアプリケーションが実稼働環境に投入される前に、適切なテストと検証を確実に実施できるようになります。

まとめ

今後の計画を立てる際には、AI とセキュリティのメリットを継続的に享受することを検討することが重要となります。これを DevOps のプロセスに導入することで、最初からセキュアで高品質なコードを提供できるようになります。生産性のボトルネックに対処し、時間を奪う要因を排除することで、エンジニアがより効率的に作業できるようになります。GitHub は、どのようなソリューションを構築しているか、どのようなフェーズを探究しているかを問わず、適切なスタート地点に立てるようにサポートします。GitHub Copilot は、開発者のエクスペリエンスの向上、セキュリティ態勢の強化、クラウドネイティブな開発でのスケーリングなどに活用できます。GitHub はあらゆる場面で皆さんをサポートします。

次のステップ

GitHub Enterprise の詳細について、または無料の試用版の利用を開始するには、https://github.com/enterprise にアクセスしてください。

著作: GITHUB