そして、ソフトウェア中心の事業は、従来のモノ中心の事業とは根本的に異なります。最大の違いは、「開発に終わりがない」という点です。

物理的な製品とは異なり、ソフトウェアは出荷された後もアップデートによって価値を高め、変化させ続けることができます。市場のニーズや社会の変化に対応し、サービスを常に更新し続けることが可能であり、むしろそれが求められる形にビジネスのルールが変わったのです。

また、世界がソフトウェア中心になるにつれて、「ソフトウェア」が指す範囲も大きく広がりました。アプリケーションだけでなく、インフラ、ネットワーク、セキュリティといった領域まで、すべてがソフトウェアによって管理される時代になっています。

DXの議論で見落とされがちですが、これほど広範囲にわたり重要性を増したソフトウェア、つまり世界の中心となりつつあるソフトウェアを支えているのは、ソフトウェアエンジニアに他なりません。

このような時代において、企業が競争を勝ち抜くためには、優秀なソフトウェアエンジニアをいかにして惹きつけ、彼ら・彼女らが生産性高く、いきいきと働ける環境を構築できるかが極めて重要な経営課題となりました。かつては外部委託（アウトソース）の対象であったエンジニアこそが、事業の最も重要な担い手となったのです。

この流れを象徴するのが、2021年に日本政府がデジタル庁を新設し、自らソフトウェアエンジニアの雇用を本格的に開始したことです。

0-3 ソフトウェア開発の中心にあるGitHub

今やソフトウェア開発に不可欠なものとなった中心的なプラットフォームが、本書のテーマであるGitHubです。

GitHubは、2008年にエンジニア同士がコードを共有し、共同作業を行うためのソーシャルネットワークとして誕生しました。その革新性から、しばしば「プログラマーのためのフェイスブック」と称されます。

GitHubは世界中の開発スタイルを根底から変え、現在のスタンダードを築き上げました。その中核機能である「Pull Request」は、コードレビューを非同期かつオンラインで円滑に行えるようにし、開発の質を飛躍的に向上させました。さらに、Pull Requestを起点としてCI/CD（継続的インテグレーション/継続的デプロイメント）を組み込むことで、それまで多大な時間を要していた開発プロセス全体を、劇的に効率化させました。

発表以来、GitHubは瞬く間にエンジニアに受け入れられ、オープンソースプロジェクトのホスティング先としてデファクトスタンダードの地位を確立します。その流れは企業にもおよび、現在ではダウ・ジョーンズ、ブルームバーグ、アメリカン航空、3M、ウォルマート、トヨタ、富士通といった世界の名だたる大企業が標準の開発プラットフォームとして採用しています。2018年にはMicrosoft社に買収され、名実ともに世界最高のソフトウェア開発基盤となっています。

さらに、GitHubの価値は開発プロセスだけに留まりません。2020年以降に世界中で定着したリモートワークという働き方とも、GitHubは深く結びついています。GitHub社自身が創業当時から10年以上にわたりリモートワークを実践しており、その経験とエッセンスが製品に色濃く反映されているのです。

ソフトウェア開発に関わるすべての作業をオンラインかつ非同期で完結できるよう設計されたGitHubは、まさに現代の働き方に最適なプラットフォームです。GitHubを使いこなすことは、効果的なリモートワークを実践することにも直結します。

0-4 多くの現場で眠っているGitHubの真価

現代のソフトウェア開発とリモートワークに不可欠なエッセンスが詰まったGitHubは、この数年で多くの開発現場に浸透しました。しかし、国内外数百社のユーザー現場を訪問した経験から、そのポテンシャルが十分に引き出されているとは言えないのが実情です。

GitHubは、開発の初期段階から運用に至るまで、ソフトウェアのライフサイクル全体をカバーする統合プラットフォームです。十数年にわたり、主要なオープンソースプロジェクトや世界の名だたる大企業で使われる中で、その機能は磨き上げられてきました。

大規模チームでの利用を前提とした高度な権限管理やユーザー管理、IT運用に求められる可用性の担保、そして脆弱性検査、IPアドレス制限、シングルサインオン（SSO）連携、監査ログの管理といった、エンタープライズレベルのセキュリティ機能も豊富に用意されています。

しかし残念ながら、GitHubを導入した開発現場の多くは、その膨大な機能のほんの一部しか活用できていません。同様に、GitHubを運用するIT部門や開発マネージャーの方々も、プロジェクト管理、セキュリティ強化、運用効率化に役立つ機能の存在に気づかず、その価値を活かしき

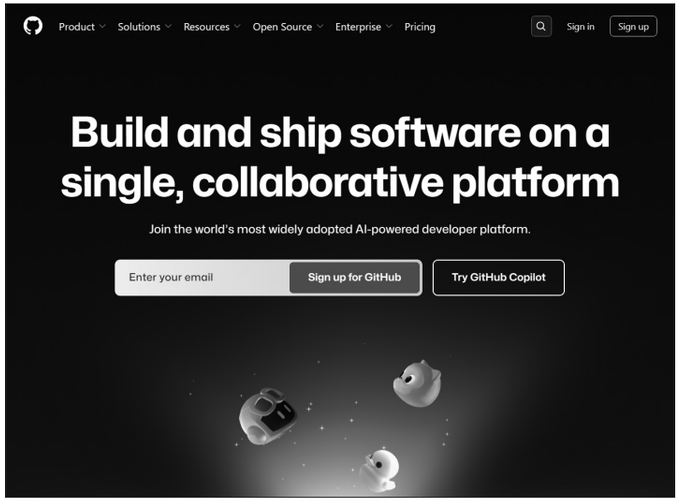
1-1 GitHubとは

ここでは、GitHubの特徴を理解してもらうために、GitHubの個人を中心とした設計思想と、チーム開発の品質と速度を劇的に向上させるPull Requestのメリットについて紹介します。

1-1-1 GitHubの思想は「プロジェクトから個人へ」

GitHub（**図1.1**）は、2008年に「プログラマーのためのフェイスブック」としてサービスを開始しました。当初はOSS（オープンソースソフトウェア）コミュニティを中心に支持を広げ、現在では日本を含む世界中の多くの企業で利用される、ソフトウェア開発に欠かせない開発プラットフォームとなっています。

図1.1 GitHubのトップページ

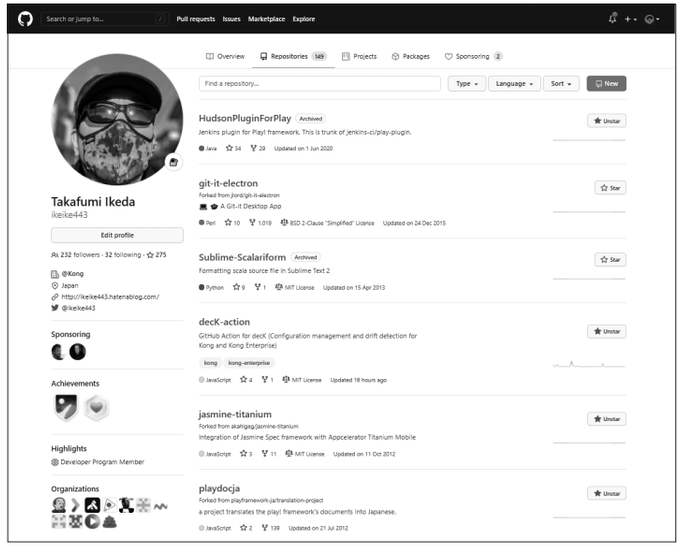


従来のバージョン管理システムやプロジェクト管理システムとの決定的な違いは、GitHubがソフトウェア開発を行う個人に焦点を当てて設計されている点です。従来、この種のツールではプロジェクトや部署が中心にあり、その下にソースコードやチームメンバーが紐づくのが一般的でした。

GitHubはこの関係を反転させ、まず個人が存在し、その個人が作成・関与するソースコードやプロジェクトが紐付くような構成になっています。

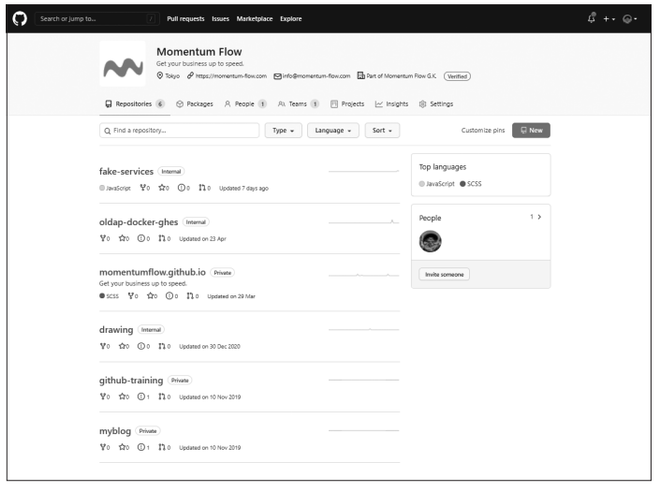
実際のユーザープロフィール画面を見ると、ユーザー自身が中心に表示され、その下にそのユーザーが管理するリポジトリ（取り組んでいるプロジェクト）が一覧で並んでいるのがわかります（**図1.2**）。

図1.2 ユーザープロフィール画面



また、プロフィールには「Organizations」という項目もあり、所属している組織がアイコンで表示されます（**図1.2**下部）。このように、プロジェクトや組織はあくまで個人の属性の一つ[注1)]として扱われ、個人と組織の間に明確な主従関係がない設計思想が見て取れるでしょう。なお、このアイコンをクリックするとOrganization画面に移動します（**図1.3**）。

図1.3 Organization画面



この思想は各リポジトリの画面にも表れています。**図1.4**を見ると、[ユーザー名]/[リポジトリ名]という形式になっていることがわかると思います。

注1） EMU（Enterprise Managed User）機能によって、企業の管理対象としてSAMLやSCIMを利用してユーザーの認証やプロビジョニングができるようになります。詳細については**第2章以降**を参照してください。

1-3-2 GitHubにおけるユーザーとOrganizationの考え方

GitHubを利用する上で、まずは「個人で使うか、チームで使うか」という視点を理解することが重要です。

GitHubは、もともと個人の開発者にフォーカスしたサービスでした。そのため、ソースコードを管理するリポジトリは、特定のプロジェクトではなく、あくまで個人に紐づくものとして設計されています。

この考え方は、リポジトリのURL構造にもはっきりと表れています。例えば、ikeike443というユーザーが持つhogeリポジトリのURLは、以下のように表されます。

https://github.com/ikeike443/hoge

このURLが示すように、hogeリポジトリの所有者はikeike443という個人であり、そのユーザーが所属する企業や組織のものではありません[注6)]。この点を理解しておくことが大切です。なお、個人アカウントでは、管理するリポジトリの公開設定（Public）と非公開設定（Private）を問わず、無料かつ無制限にリポジトリの作成が可能です。

一方、チームでの開発や管理のために、GitHubにはOrganizationという概念があります（**図1.13**）。これは特定の個人ではなく、企業、オープンソースプロジェクト、NPO、教育機関といった組織を指します。

Organizationを利用すると、リポジトリを組織に帰属させることができます。例えば、著名なフレームワークであるRuby on RailsのURLは以下のようになっています。

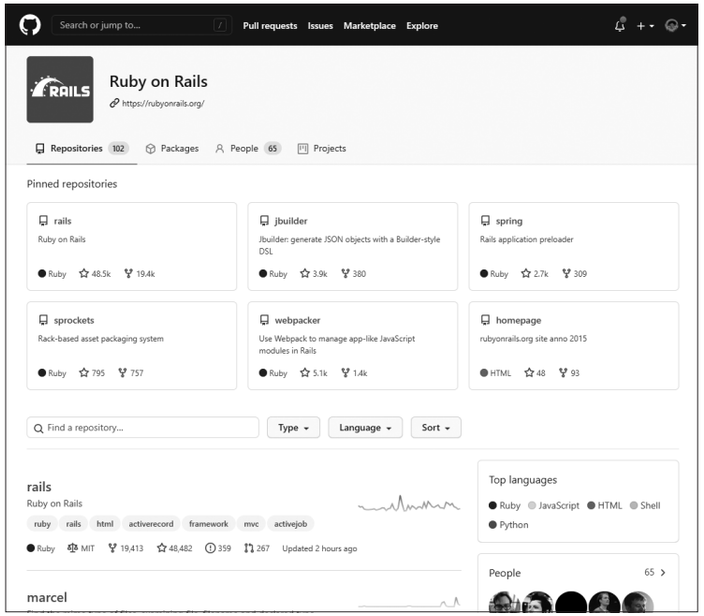
https://github.com/rails/rails

この場合、railsというリポジトリがrailsというOrganizationに所属していることが分かります。

Organizationを利用する大きなメリットは、組織内のリポジトリやメンバーに対して、チーム単位での細かい権限管理が可能になることです。Organizationでも個人の場合と同様に、公開・非公開を問わず、無料でかつ無制限にリポジトリの作成が可能です。

このように、GitHubにはまず個人（ユーザー）と組織（Organization）という、コードの所有者に基づいた2つの主要な概念が存在します。そして、これらの利用形態と、利用できる機能によって異なる有償・無償のプランは、それぞれ独立した考え方であり、直接関係するものではありません。

図1.13 Organizationの画面、リポジトリ



1-3-3 GitHubの3つのサービス形態

GitHubのサービスは、その提供形態によって大きくクラウド型とホスト型の2種類に分類されます。多くの方が利用しているGitHub.comは、GitHub社が管理・運営するクラウド型のSaaS（Software as a Service）です。

一方、より高度なセキュリティ、コンプライアンス、管理機能が求められる大企業向けには、本書で解説するGitHub Enterpriseという製品が用意されています。このGitHub Enterpriseには2つの選択肢があり、一つはGitHub.comと同じインフラ上でEnterprise向けの追加機能を利用できるクラウド型のGitHub Enterprise Cloud（GHEC）、もう一つは、自社のサーバーなどにインス

注6） これについては、GitHubの利用規約に明記されています。
https://docs.github.com/ja/site-policy/github-terms/github-terms-of-service
ただし、本書で扱うGitHub Enterprise Serverの場合はこの限りではありません。

合はオンプレミス固有のモードなどの要素が影響してくるためです。

Organization と Team を適切に設定することで、権限管理を柔軟に行うことができます。今回のケーススタディのように、社員ステータス（正社員か否かなど）によってアクセス権限を分けたい場合は、「正社員 Team」などの Team を作成して管理すれば良いのです。

さらに細かい権限管理を行いたい場合は、GitHub の権限ポリシーについての理解を深めることで、より詳細にかつ正確に設定できるようになります。GitHub の権限ポリシーについては、**第3章**を参照してください。

CASE.2 Team を階層構造にして効率的に権限管理したい

ケーススタディ

新規 Team を作成する際、既存 Team の権限設定を利用しながら独自の権限も設定し、複雑な組織構造に対応するとともに、柔軟な権限設定を行いたい。

最適解へのポイント

- Organization 配下のリポジトリに対して、どの Team がどのような権限を持つかを設定して、柔軟な権限設定を可能にする
- Team に親子関係を持たせて、親 Team の権限設定を子 Team が自動的に継承し、さらに子 Team 側に独自の権限を設定することで、複雑な組織構造に合わせた権限管理を行う

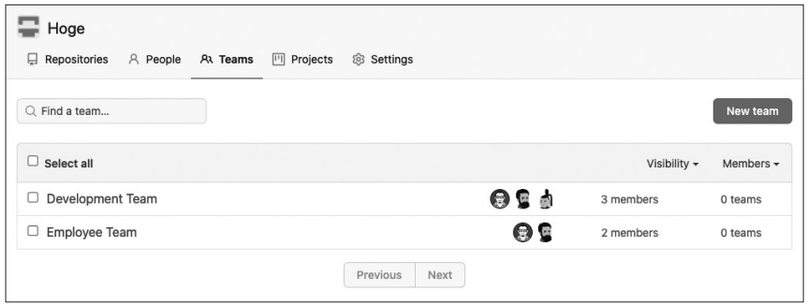
解説

GitHub の Team 機能は、リポジトリの権限管理に非常に役立ちます。実は、あまり知られていませんが、Team は親子関係のような階層構造を持つことができます。

権限管理を複雑化させる要因

たとえば、**図2.2**のように、正社員 Team（Employee Team）と開発 Team（Development Team）がすでに作成されているとします。

図2.2 既存の Team

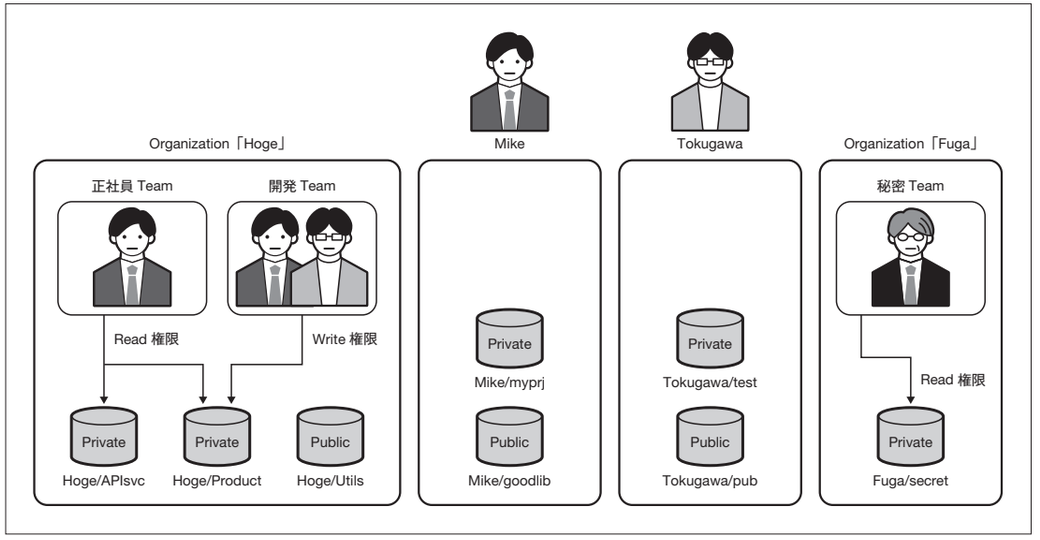


Team では、リポジトリだけでなく、プロジェクトなど、さまざまな機能の権限を管理できます。ここでは、Team の階層構造を使うことによって、これらのアクセス権限をさらに細かく、柔軟に管理していく方法について解説します。

具体的なシナリオで解説を進めるために、Case.1 で示した GitHub の構成とアクセス管理図を再掲します（**図2.3**）。

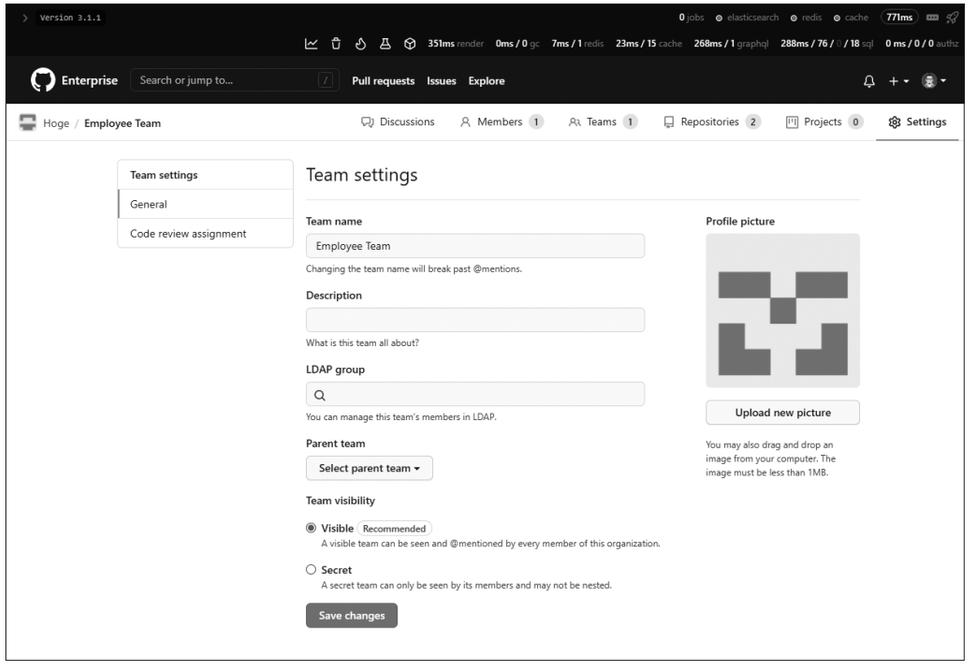
図2.3 では、正社員 Team が Hoge/APIsvc リポジトリと Hoge/Product リポジトリに Read 権限を持ち、開発 Team が Hoge/Product リポジトリに Write 権限を持つ構成になっています。これは、正社員ではない Tokugawa 氏にも Product への Write 権限を与えるための工夫でした。

図2.3 Organization と、Team、リポジトリの関係例（再掲）



自動的にGHESのTeamからも外されます。この仕組みによって、LDAPグループのメンバー構成をGHESのTeamに自動的に反映できます。

図2.24　LDAP group



LDAP Groupフィールドで、対象のGroupの先頭何文字かを入力することで検索による補完が効き、効率的な設定が可能です(**図2.25**)。もし、目的のGroupが出てこない場合は、Domain baseやRestricted user groupsの設定を見直してください。目的のGroupが検索範囲に含まれていない可能性があります。

図2.25　LDAP Groupの検索欄

LDAP Groupでは、LDAPのGroupのみ指定可能なことに注意してください。本来であれば組織単位(OU)で指定したいところですが、サポートされていないため、対象Teamごとに LDAPグループを作成する必要があります[注8)]。

この設定を行うと、当該Groupのメンバーが、自動的にそのTeamメンバーとして同期されるようになります。ただし、ユーザーのプロビジョニングが完了していない状態では、Team Syncは動作しません。まずサイト管理者が手動でGHESにユーザーを作成するか(Site Adminページから「Create」)、ユーザー自身がGHESにサインインし、プロビジョニングを完了させる必要があります。

また、Team Sync設定後は、GHESの画面からTeamメンバーの追加や削除などの操作は行えなくなります。以降は、そのTeamメンバーはLDAP情報が正しいものとして扱われるため、LDAP上でメンバーの追加・削除など操作を行うことで、GHESのTeamに反映されるようになることを念頭に設定を行ってください。

CASE.4 SAML/SCIMでユーザー管理を自動化したい

ケーススタディ

大企業では、組織変更や人事異動がひんぱんに行われるので、それに伴うGitHubの設定変更は非常に手間がかかってしまう。セキュリティもきちんと強化して、さらにユーザー側にとってもアクセスしやすいような環境を整えたい。

最適解へのポイント

- IdPとGHES/GHEC間のSAML設定を正しく行って、リカバリーコードを安全に保管する
- Gitの操作にはPersonal Access TokenまたはSSHキーのSSO認可を行う
- SCIMを設定してユーザーとアクセス権の自動同期を有効にする

注8)　GHESがLDAPのOUを直接サポートせず、代わりにTeamごとにLDAPのGroupを作ることを推奨していることから、GHESはLDAPサーバー全体の組織図を取り込んで、それに基づいてTeamを自動的に構築することを想定していないようです。むしろ、LDAPとの連携においては、Domain baseとRestricted Groupで対象範囲を限定した上で、組織図とは別に、GHESのTeamごとにGroupを設定するという設計になっていると考えると、わかりやすいでしょう。

管理を考えていくと良いでしょう。

CASE.6 私用メールアドレスなどに通知が漏れるのを防ぎたい

ケーススタディ

業務でGitHubを利用する際、私用のメールアドレスで登録したGitHubアカウントをそのまま使われると、業務関連の通知が私用のメールアドレスに漏れるというセキュリティ上の懸念が出てしまうので対策したい。

最適解へのポイント

- 業務専用のアカウントを企業のメールアドレスで作成する
- 業務に関するやり取りをGHESに限定し、ID基盤と連携して勝手にメールアドレスを追加するのを防ぐ
- (GHECのみ)メール通知を行うドメインを限定する

解説

GitHubを業務で利用する際、社員の私用メールアドレスへ業務に関連した通知が送信されてしまうというセキュリティ上の懸念が出てきます。

GitHubはもともと個人開発者向けのサービスとして始まっており、すでに作成しているGitHubアカウントを業務でも利用するケースは多々存在するためです。これによって、業務に関連する通知メールが私用メールアドレスに届いてしまうリスクが生じます。

この問題に対処する方法として以下の3つがあります。

①仕事用とプライベート用でアカウントを分ける
②GHESを利用する
③Verified Domain機能で通知先を制限する

仕事用とプライベート用でアカウントを分ける

最初に思いつくのは、業務専用のGitHubアカウントを新規作成することでしょう。私用メールアドレスとは別に会社のメールアドレスで新たにアカウントを作成し、業務ではそれを使ってもらうようにします。

以前のGitHubでは複数アカウントは許可されていなかったのですが、2025年10月現在では複数アカウントを作って、切り替えることができるようになっています。

GHESを利用する

また、解決策の一つとしてGHESを利用する方法もあります。ただし、単にGHESを導入しただけでは、社員が自分のプロフィールに私用のメールアドレスを追加できるため、通知が漏れるリスクは残ったままです。

このリスクを解消するには、GHESとLDAP/SAMLなどのID基盤を組み合わせます。メールアドレスは会社側で一元管理され、社員が勝手に私用メールアドレスを追加することが不可能になります。セキュリティを最大限高めるには、非常に堅固で有効な選択肢です。

Verified Domain機能で通知先を制限する

GHECまたはGHESを利用している場合は、メール通知の送信先を会社が許可したドメインだけに強制することが可能です。

Organization画面のSettingsタブで、「Verified & approved domains」を開きます(**図2.54**)。画面右上の「Add a domain」をクリックして、自社のメールドメインを登録して認証を行います。

図2.54 Verified & approved domainsメニュー

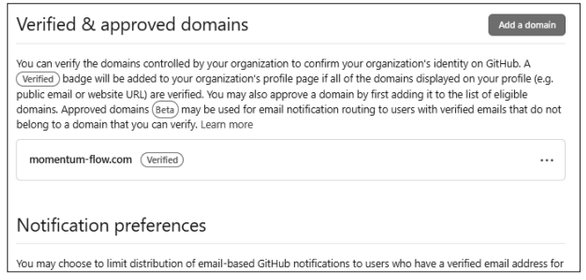


図2.55 認証バッジが表示される

認証完了後は、Organizationのプロフィール画面に認証バッジが表示されるようになります(**図2.55**)。

監査ログを閲覧するには、Enterpriseの管理画面から「Settings」－「Audit log」を選択します。この画面では、Organizationレベルよりも広い範囲、つまり当該Enterpriseに属する複数のOrganizationを横断した監査ログの検索・取得が可能です。

検索構文の書き方は、Organizationレベルと同じですが、Enterpriseレベルでは、より重要なセキュリティ関連の監査ログが記録される、という違いがあります。また、GHECではOrganizationレベルと同様に、ログをファイルとしてエクスポートするボタンが2つ用意されています。

監査ログの取得方法

このように監査ログは画面上での確認やエクスポートが可能ですが、何かが発生するたびに手動でエクスポートするのは手間がかかりますし、保存期間が90日（GHECは180日）では長期保存のニーズにも応えられません。

これらの課題を解決するには、監査ログをファイルに書き出して別の場所にアーカイブしたり、データ分析基盤や監視ツールに直接転送したりする方法が一般的です。

GitHubの場合、監査ログの取得方法はGHECとGHESの場合とで複数それぞれ用意されています。

GHECの場合、まず先ほど紹介したエクスポートボタンを使ってその場でファイルに書き出すことができます。また、GHECまたはGHESの双方において、GraphQLとRESTの二方式のAPIでそれぞれAudit logを取得できるようにしています。

加えて、GHECおよびGHESの双方において、特定監視ツールへのログストリーミング機能が搭載されています。

次にGHES固有の強力な方法として、汎用的なログ転送機能が用意されています。GHESは内部でsyslog-ngを利用しており、GHES内部で出力しているあらゆるログをsyslog-ngで外部の指定されたエンドポイントにストリーミングすることができます。

このログの中にAudit logも含まれているという寸法です。Audit logの他に、HAProxyのアクセスログ、Git LFSやGitHub Issueで取り扱っているファイル保存のログ、認証のログなど、さまざまなシステムログをすべて転送することができます。

◆ ◆ ◆

このように、GHECとGHESそれぞれ、さまざまな方法で監査ログを取得する方法が用意されていることがわかったかと思います。

CASE.9 Gitの操作についてログを確認したい

ケーススタディ

情報システム部門は、セキュリティチームからGit操作ログの提出を求められている。特に、重要なリポジトリへの強制プッシュ（force push）といった操作は厳密に監査する必要がある。GHECとGHESの両環境で、これらの監査要件を満たすログを効率的に収集・監視したい。

最適解へのポイント

- GHECでは、Audit Logのストリーミング機能を使い、監査ログを外部システムへ自動転送することで、リアルタイム監視とログの長期保存を実現できる
- GHESでは、標準のログ転送機能でbabeldログを取得して、誰がどのIPアドレスからリポジトリを操作したかといった詳細なGit利用履歴を確認できる

解説

監査目的でGit操作についてログを取得するには、GHECとGHESの場合で複数の方法が用意されています。

GHECの場合

GHECの場合、前述の「Export Git events」ボタン（**Case.8参照**）を利用してその場でファイルに書き出すのが一番簡単な方法です。なお、他の監査ログの保存期間が180日間であるのに対し、Gitイベントは7日間と非常に短いので注意してください。

図2.63は、Gitエクスポートを実行すると取得できる情報の例です。git.fetchやgit.pushなど

3-1 GitHubを構成するプロダクト

3-1-1 多種多様なプロダクト

GitHubは非常に多機能で、個人でプログラミングを楽しむ人から、大企業のシステム開発チームでシステム開発に携わるプログラマー、プロジェクトマネージャー、品質管理担当者など、さまざまなユーザーペルソナを想定した巨大でかつ複雑な開発プラットフォームです。しかし、その複雑さを感じさせないように、必要な機能を直感的にすぐに使い始められるのがGitHubの大きな利点です。

一方、企業でGitHubを管理する担当者にとっては、必ずしも直感的で使いやすいシステムとは言えません。さまざまなユーザーペルソナのニーズを満たすために実装された機能やその構造のすべてを少数の担当者で把握するには複雑すぎるためです。

本章では、GitHubの運用担当者や導入を検討している責任者に向けて、GitHubの基本的な仕組みを理解し、最適な運用方法を構築するための基礎知識を提供します。

具体的には、GitHubの主要な機能や利用できるプランについて説明した後、特にGitHub Enterpriseプラン（以下、Enterpriseプラン）に焦点を当て、実際の運用に必要な権限管理の基本や構造について解説します。

3-1-2 本書におけるプロダクトの定義

本書においては、プロダクトの定義を「その単位で運用について考慮する必要があったり、またその単位で料金が別途発生するもの」としています。この定義には当てはまらないが、別の機能を持った機能群をモジュールと定義し、プロダクトの下に位置付けています。

なお、これらの定義や分類は著者による見解で、必ずしもGitHub社の見解と一致しているわけではないことをご承知おきください。

表3.1に挙げたプロダクトは、無料で利用可能なものから、サブスクリプション加入必須のものまでさまざまです。機能によってはサブスクリプション加入と別で従量課金が発生するものもあります。サブスクリプションについても、Teamプランで利用可能なもの、Enterpriseプラン

表3.1 GitHubを構成する主なプロダクト

カテゴリ	プロダクト・機能	説明
リポジトリ・コラボレーション	Repositories	ソースコードやプロジェクトファイルをバージョン管理するための中心的な場所（リポジトリ）
	Issues	タスクやバグを追跡して課題を管理する機能
	Projects	Issueをカンバン形式で視覚的に管理する機能
	Discussions	コードとは直接関係ないコミュニケーションを行うための掲示板機能
	Pages	静的なWebサイトをホスティングする機能
	Gist	断片的にコードや文章を共有できる機能
	Organizations・Teams	企業や組織でGitHubを利用するための管理機能。Teamによってリポジトリへのアクセス権限などを柔軟に設定することが可能
	Git LFS	数十MB以上の巨大なファイルを効率的に管理するための拡張機能
CI/CD・パッケージ管理	GitHub Packages	Dockerイメージやnpmパッケージなど、さまざまな形式のソフトウェアパッケージをホスト・管理できる機能
	GitHub Actions	CI/CDを実現するワークフロー自動化機能
開発ツール	Codespaces	クラウド上で完結する開発環境。ブラウザから直接コーディング、ビルド、デバッグが可能
	クライアントツール	GUIアプリのGitHub Desktop、コマンドラインツールのGitHub CLI、エディタの拡張機能など
	API	REST APIやGraphQL APIを通じて、GitHubの機能を外部のアプリケーションと連携させることが可能
AI開発者支援	GitHub Copilot	AIがコードの補完、提案、生成を行うペアプログラマー。チャット形式で質問やコードレビューも可能
セキュリティ	Secret scanning	コード内に誤ってコミットされたAPIキーや認証情報などの機密情報を検知し、漏洩を未然に防ぐ機能
	Dependency Review	Pull Request内の依存関係の変更をチェックし、脆弱性を持つライブラリの追加をブロックする機能
	Dependabot	プロジェクトが利用しているライブラリの依存関係を監視し、脆弱性が発見された際に自動で修正Pull Requestを作成する機能
	Code scanning	コードを解析し、潜在的な脆弱性やコーディングエラーを自動で検出する静的解析エンジン
コミュニティとその他	Sponsors	オープンソース開発者やプロジェクトに対して、個人や企業が資金的な支援を行えるスポンサーシップ機能
	Search	GitHub上のすべてのPublicリポジトリを横断してコードやIssueの検索が可能な全文検索機能

Audit log

「Audit log」では、Organizationに追加された際のログを確認することができます。

Organization affiliationsメニュー

「Organization affiliations」では、当該ユーザーがアクセス権を持っているOrganizationの一覧を表示し、それぞれどのようなロールで、いくつのTeamに参加しているのかなどの確認や、当該Organizationからの削除などが可能です。

Organization securityメニュー

「Organization security」では、各Organizationにおけるセキュリティの設定状況を確認することができます。

Collaborating repositoriesメニュー

「Collaborating repositories」では、このユーザーがOrganizationの正規メンバーとしてではなく、外部協力者（Outside Collaborator）として個別に招待されたリポジトリの一覧を確認できます。

Collaborationタブのその他のメニュー

Collaborationタブのその他のメニューについては**表5.3**にまとめています。

表5.3　Contentタブ（その他のメニュー）

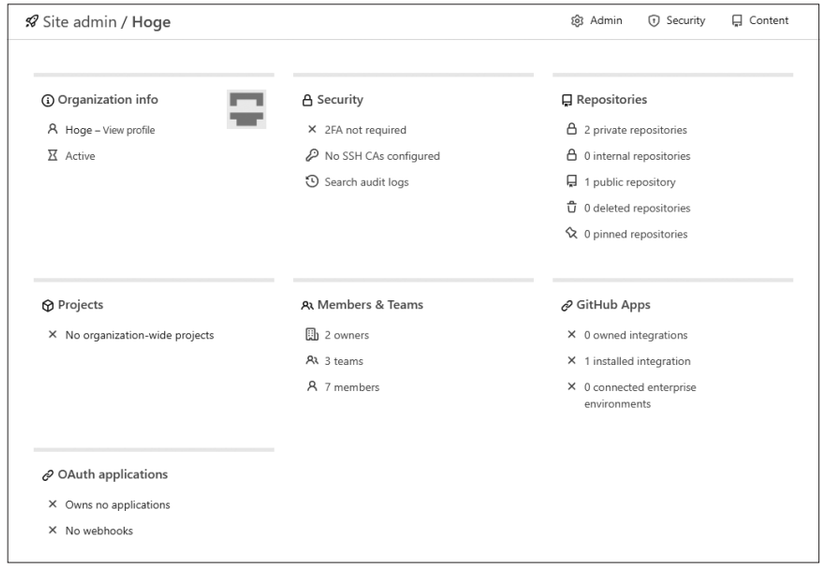
メニュー	説明
Watched&Ignored repositories	ウォッチまたは無視しているリポジトリの一覧を確認する
Starred	スターが付いているリポジトリを確認する
Contributing repositories	当該ユーザーが実際にコミットを Push しているリポジトリを確認する
Recent comments	コメントを確認する

5-1-9 Organization詳細のタブメニュー

次にSite AdminのOrganization画面について見ていきましょう（**図5.18**）。

Organizationのトップページの右上部には、「Admin」「Security」「Content」タブがあります。

図5.18　Organizationの詳細画面



5-1-10 Adminタブ

Organization詳細のAdminタブのメニューは、ユーザー詳細のAdminメニューと共通部分が多いため、本項ではOrganization詳細についてのポイントをかんたんに触れていきます。

Overviewメニュー

「Overview」では、当該Organizationの設定状況などを一覧で確認できます（**図5.19**）。

Adminメニュー

「Admin」にある設定項目（Contribution graphリビルド、Organization名の変更やForce pushなど）は、ユーザー詳細のものとほぼ同じです。一番の違いは、ここでの設定がユーザー個人ではなく、Organization全体に適用される点にあります。

Emailsメニュー

「Emails」では、当該Organizationの連絡先メールアドレスを設定できます。Emailsに続く「Avatars」「Search」「Database」はユーザー詳細のものと同じです（**5-1-5参照**）。Retired namespaces

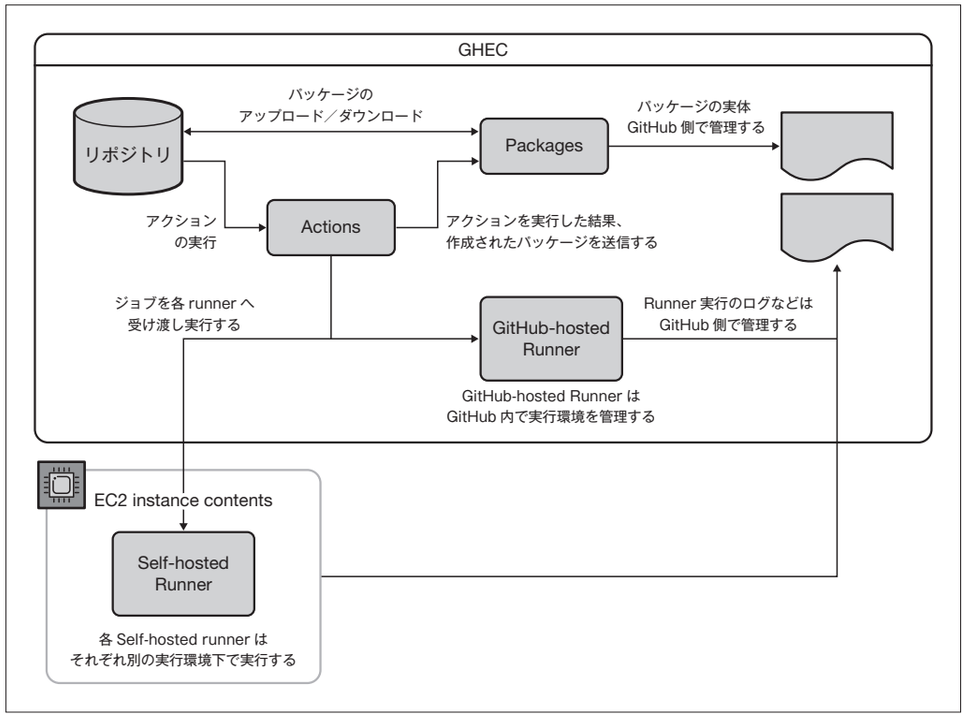
GHES本体やRunnerの実行環境、Amazon S3などストレージサービスは、同一社内ネットワーク内に置くことが望ましいですが、仕様としては必須要件になっていません。理論上は**図6.1**の構成要素をインターネット越しで構築することもできます。

ただし、AWSを利用する場合は、GHES本体をEC2上に置くことになるため、すべての構成要素を同一VPC上で構築することが可能です。通常はそのような構成にするのが望ましいでしょう。

GHECにおけるActions/Packagesのシステム概要

GHECにおけるActions/Packagesのシステム概要図は、**図6.2**の通りです。

図6.2　GHECにおけるActions/Packagesのシステム概要図



GHECの場合は、SaaSとしてすべての実行環境が提供されているので、GHESのように別途Amazon S3などを用意する必要はなく、すぐに利用可能です。

Runnerについても、特別な要件がなければ、GitHubがOSごとに用意・管理しているGitHub-hosted Runner（**6-1-6参照**）を利用できます。

ただし、**図6.2**にあるとおり、要件に応じてGHECでSelf-hosted Runnerが利用できるので、ユーザーが用意した別環境でも実行させることが可能です。

6-1-6　Runner（ランナー）

Actionsの Workflowで定義されたジョブを実行するコンピューティング環境をRunnerと呼びます。Runnerには、GitHub-hosted RunnerとSelf-hosted Runnerの2種類があります。

GitHub-hosted Runner

GitHub-hosted Runnerは、GitHubが管理しているRunnerです。利用者がその存在を意識する必要はなく、Workflowファイルの中で実行環境を指定すれば、自動的にGitHub-hosted Runnerが動作し、適切な処理を行ってくれます。

GitHub-hosted RunnerはGHECのみで使用可能で、プランに含まれる無料枠を超えた分については、実行時間に応じて従量課金が発生します。

Self-hosted Runner

Self-hosted Runnerはユーザー自身で実行環境を用意して管理するRunnerです。Runnerのプログラム自体は、GitHub社からオープンソースとして提供され、一度セットアップすれば、基本的に自動更新されます。

GitHub.comやGHEC、GHESで利用可能ですが、GHESの場合はSelf-hosted Runnerが唯一の選択肢となります。GitHub-hosted Runnerと異なり、Self-hosted Runnerの利用に従量課金はありませんが、その実行環境の維持・管理コストはユーザーが負担します。Self-hosted Runnerの実行環境に要求されるスペックなどは、公式ドキュメント[注4)]を参照してください。

Self-hosted Runnerの集中管理

Self-hosted Runnerを導入すると、社内ネットワークにさまざまなRunnerが入り乱れる可能性があり、セキュリティやリソース管理上の問題が発生しがちです。そこで、情報システム部門などがGHES管理担当を配置し、可能な限りRunnerを一元的に管理する体制を組んだ方がよいで

注4）　https://docs.github.com/ja/actions/concepts/runners/self-hosted-runners