“The book” on GitHub Enterprise Cloud Adoption

WRITTEN BY GITHUB WITH ♥

What's inside

Authors:
Philip Holleran Field CTO, GitHub
Kevin Alwell Principal Engineer, GitHub

Introduction

Document purpose

This document provides an Enterprise playbook for technology decision makers, developer tools teams and security specialists responsible for understanding the GitHub Enterprise Cloud platform, architecting the environment for their business, creating and executing a migration roadmap, and making a business case demonstrating the value of developing software on the GitHub platform.

What is GitHub?

Understanding GitHub

GitHub is the home of 100 million software developers and counting. It's where builders collaborate on software from Idea to Production. The platform is uniquely fortunate to anchor the Open Source community, including some of the world's most influential projects. Enterprises operating in the platform realize tremendous value from standardizing on such a ubiquitous, developer centric software delivery platform, due in no small part to platform services being informed by data from Public repositories. Historically GitHub has been hyper focused on the developer experience for git based Source Code Management. Since the Microsoft acquisition in 2018 the platform has evolved from a leading SCM tool to an Enterprise end to end software delivery platform inclusive of native best in class SCM, Planning and Tracking tools, CI/CD, Application Security, IDE as a Service and AI pair programming. Core to our strategy is support for teams that seek to employ the Best of Breed approach to the developer toolchain. The community of developers and Enterprises attract modern service providers who build world class integrations to support their workflows (ex: Jira, Jenkins, Azure DevOps).

Best in class Developer Experience

Developers know and love working with GitHub. This accelerates time to productivity and improves overall satisfaction leading to talent acquisition and retention benefits.

A fully integrated developer toolchain prevents unnecessary context switching, keeping developers in their flow delivering software. According to our latest Forrester report, after using GHEC and GHAS for three years, composite organizations see > 22% productivity gains overall.

Code Reuse and collaboration are prolific within the platform, encouraged by an accommodating architecture and default innersource philosophy.

Largest Connected Developer Community

- Largest community of developers in the world, working in one place generating millions of lines of open source software that can be reused.
- Enterprise customers derive value from the OSS community in numerous ways including through well maintained Service Provider integrations, Data driven services like Dependabot and Copilot.
- Ubiquity of GitHub drives familiarity, impacting developer productivity and happiness.
- Access to reusable components already built by the community
- Ability to share open innovation to help global causes

Integrated, Developer Centric Application Security

- Application Security is native within GitHub through Code, Secret and Dependency Scanning.
- Result quality is a central focus, reducing friction and improving mean time to remediation.
- Integrated AppSec maintains the developer flow preventing a productivity killing, disjointed experience.
- Niche 3rd party integrations are supported through a first class integration workflow.

Stable, Innovating partner for the long term

- In 2018, GitHub became a Microsoft company. Microsoft has since communicated that GitHub is their preferred developer tools platform.
- Since the Microsoft acquisition GitHub has hired thousands of Engineers that continue to mature core services resulting in deeper integration, greater value and tool consolidation.
- GitHub continues to differentiate through services like GitHub Advanced Security, Copilot and Codespaces.
- GitHub continues to demonstrate customer centricity when developing and delivering on the Product Roadmap.

Capabilities

Source control and collaboration

A GitHub repository contains all of your project's files and each file's revision history tracked through the git version control system. You can discuss and manage your project's work within the repository. Additionally, the Repository interface provides access to other native developer tools such as GitHub Advanced Security, Actions, Copilot, CodeSpaces and Projects.

Security

GitHub Advanced Security is comprised of developer centric, best in class, native Application Security tools. GHAS includes Code, Secret and OSS dependency scanning.

Automation

GitHub Actions is a native continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

Planning

A GitHub Project is an adaptable project planning board that integrates with your repositories issues and pull requests to help you plan and track your work effectively.

Developer environments

GitHub CodeSpaces is a native, cloud hosted IDE configured as code, designed to accelerate time to productivity and reduce the overhead of maintaining a local IDE for application development.

AI Code Completion

GitHub Copilot uses the OpenAI Codex to suggest code and entire functions in real-time, right from your editor. Research has found **GitHub Copilot helps developers code 30% faster**, focus on solving bigger problems, stay in the flow longer, and feel more fulfilled with their work.

Discussions

GitHub Discussions is a collaborative communication forum for the community around an open source or internal project. Community members can ask and answer questions, share updates, have open-ended conversations, and follow along on decisions affecting the community's way of working.

Adopting GitHub

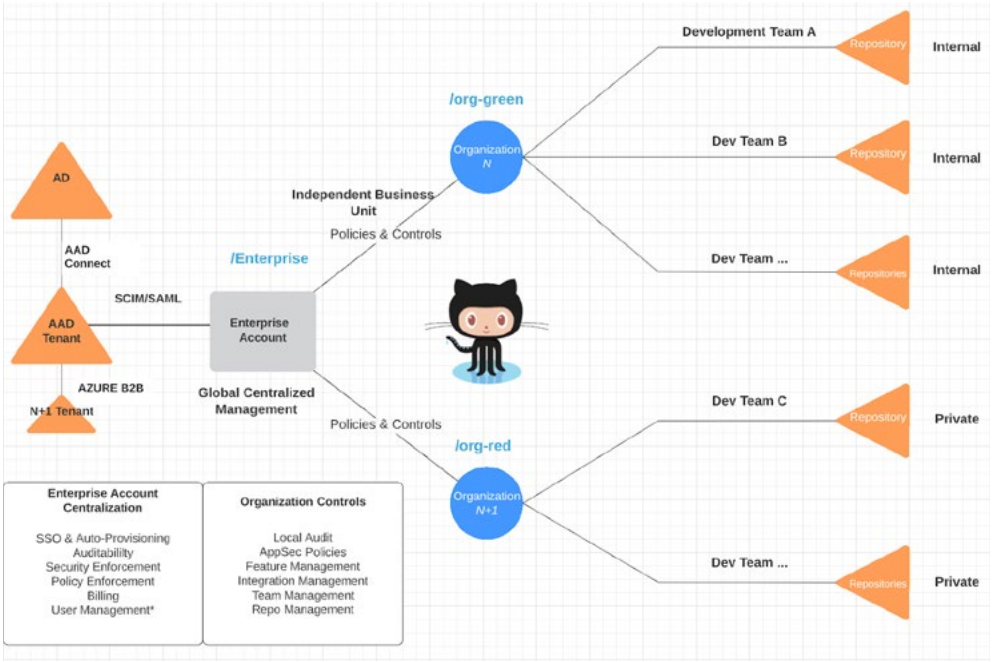
Cloud first

As a SaaS platform, GitHub Enterprise Cloud is the most operationally efficient deployment option for Organizations operating at scale on GitHub. In addition to its ability to scale to support Enterprises with more than 50,000 developers, the platform provides resilience and disaster recovery out of the box. Modern Enterprises are particularly attracted to GHEC because their developers will gain access to the latest tools available on the platform, some of which have no path to support on the hosted solution such as CodeSpaces. Feature availability is a major competitive advantage, as is time to market. On Cloud, Enterprises will gain access to features months and sometimes quarters before they ship to server.

Thinking about structure

Organization structures are constantly evolving, especially in acquisitive and/or technology driven environments. Your GitHub architecture should reflect your intention to promote a collaborative developer experience that respects the complex and dynamic nature of your business.

Platform Architecture (structure your environment)



Constructs

Enterprise Account

The Enterprise Account is the highest level, logically isolated construct within GitHub Enterprise Cloud. It is what administrators use to manage how GitHub interfaces with internal business systems like your identity provider (IdP) and log management system. The Enterprise Account also provides the ability to define, and enforce, policies governing the use of GitHub resources and capabilities. For GitHub Advanced Security users the Enterprise Account also receives AppSec insight derived from Organizations operating within its confines.

Organization

Enterprises consist of one or more organizations, to which users are added. Organizations are the "owners" of shared repositories, discussions, and projects. They let administrators set more granular policies on repositories belonging to the organization and user behavior within the organization. Policies not enforced at the enterprise level are distributed out to individual organizations.

Organizations also serve as a roll-up mechanism for reporting. GitHub Advanced Security provides a summary of the security status of repositories in an organization. Consumption-based services, such as GitHub Actions and Codespaces, are reported at both the repository and organization level. Spending limits on these services can be set on a per-organization basis.

Repository

Repositories are your application source code. In addition to source code, repositories are the way users access application security metrics, CI/CD workflows, and other day to day activities for developers.

This is the primary construct developers interface with on GitHub. There are three visibility types for repositories: Public, Internal, and Private.

- Public repositories are visible to the world and are primarily used for Open Source content.
- Internal repositories are public to members of your enterprise account.
- Private repositories are only visible to admins and individuals or teams who have been given explicit access.

Team

GitHub Teams group users of common projects or specialized skills, they are often the mechanism for providing role based access to collections of repositories.

User

GitHub Users are mapped to individual people. They can be added to Organizations as Members and repository Teams as collaborators. Two user models exist:

- Standard user accounts
- Enterprise Managed Users (EMUs)

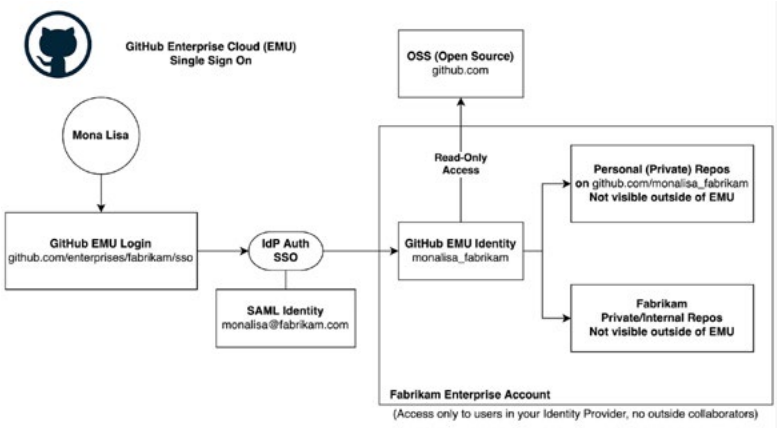
GitHub.com Standard User

Standard Users on GitHub.com are owned and operated by individuals and are designed to follow that individual as they join, contribute and leave Organizations throughout their life. These Users can actively participate in the OSS community.

Standard GitHub Users can be invited to join Organizations as contributors gated by SAML 20.0 SSO. SCIM governs the workplace identity association and authorizes or terminates access automatically. Teams can also be associated with IDP groups via Team Sync to keep GitHub Teams in sync with AD Group membership.

Enterprise Managed User

Enterprise Owned user accounts on GitHub.com that operate exclusively within the confines of an Enterprise account on proprietary source code. These user accounts are created, updated and deleted by the Enterprise. EMUs are managed by the Enterprise IDP, namely Azure Active Directory or Okta. More Information.



Enterprise Managed users provide an Enterprise Owned and managed construct that has limited write authorization to repositories outside the Enterprise account. Their handles are defined through the Enterprise IDP and all access is subject to Conditional Access Policies including IP restrictions. EMUs are provisioned through SCIM integration, synced via IdP group membership with GitHub Teams, subject to Enterprise SSO and conforming to enterprise naming conventions.

How (opinionated)

GitHub Enterprise Cloud provides a number of flexible configuration options, allowing each business to configure the platform to best meet their unique needs. While there is no single "best" way to implement GitHub Enterprise Cloud, there are some common implementation patterns and steps you should carefully consider.

The following sections discuss recommended configuration and implementation of your:

- Enterprise Account
- Organization(s) and their
  - Repositories
  - Teams
- User Accounts

One Enterprise to rule all (internal) things

Successful GitHub Enterprise Cloud implementations begin with establishing a single enterprise account for all internal work. This enterprise will govern policy across your GitHub organizations, users, and teams, and be your single portal for managing billing

Deciding on a user model

When creating the enterprise, you'll need to make one key decision - which user model works best for your company: Enterprise Managed, or Standard? The details of each were covered in the previous section.

Ultimately, if you need full control of your developers' accounts and a firm separation between the open source community and your enterprise code - EMUs are for you. However, if your developers will be regularly building and maintaining open source code as a part of their daily work the Standard model may be best.

Global configurations

Once your enterprise is established it's time to configure settings and policies that apply to all development work. Configurations and policies not applied at the enterprise level are managed in each organization. This lets you manage your desired balance between centralized and distributed administration.

Authentication

The first setting to configure is authentication. A few minor differences exist between Enterprise Managed Users and Standard Users, so we'll cover each of them in its own section.

Enterprise Managed Users

When using Enterprise Managed Users a single Identity Provider (IdP) is configured at the enterprise level. Supported IdPs include Azure Active Directory (AAD) and Okta. If you need to support users from more than one IdP, you will need to configure either AAD or Okta to federate your various identity providers and expose a single store with which GitHub Enterprise Cloud will integrate.

EMUs and outside collaborators

Every user account in an EMU enterprise must be mapped to an identity in your IdP. Collaborators, consultants, and other outside parties will require an identity in your IdP. These identities do not have to exist in your enterprise identity store (LDAP, AD). Azure AD includes the concept of a guest user and both Azure AD and Okta provide support for multiple user stores.

EMUs and Conditional Access Policies

If you use Azure AD, you can configure GitHub to use OpenID Connect (OIDC) to manage authentication with your IdP. Doing so adds support for Azure AD's conditional access policies.

Standard Users

When using standard users you have the option to either:

- Configure a single SAML 2.0+ provider at the enterprise level, or
- Configure each organization with a SAML 2.0+ provider

Most customers using standard users opt for a single, enterprise-level IDP integration. However, if your business has several, disparate identity systems, the organization-level configuration may work best. This multi-IdP approach is most commonly used in cases where a business operates as a set of very-loosely held portfolio companies, each with independent IT departments, tools, and processes.

Audit log

GitHub audits events triggered by activities in your enterprise as well as git events, or those specifically associated with the pushing/pulling/cloning of code. GitHub maintains non-git logs for a configurable period up to six months, while git events are

retained for seven days. Data from the logs is available to administrators within the GitHub UI, via the API, and via streaming.

Customers should configure audit log streaming within GitHub to ensure all data captured is held within their preferred log management system and retained in accordance with existing policies. GitHub currently supports native integration with:

- Azure Event Hubs
- Datadog
- Splunk

as well as the ability to directly write to:

- AWS S3
- Azure Blob Storage
- Google Cloud Storage

for ingestion into your tool of choice. Some customers use this data for anomaly detection via tools like Defender for DevOps and the GitHub App for Splunk.

Policies

The policies pane in your GitHub Enterprise account enables the definition, and enforcement, of a variety of policies across your enterprise, broken down by topic:

- Repositories
- Copilot
- Actions
- Projects
- Teams
- Organizations
- Code security and analysis

Companies using Enterprise Managed Users should note that, unless explicitly stated, these policies apply over Organizations but do not apply to activity in personal namespaces.

Repositories

Policies in this section govern repository creation and management.

Base permissions represent the default set of permissions (if any) granted to all members of the enterprise. In most cases this will either be “no permission” or “no policy” in order to allow for limited access to particularly sensitive repositories. A number of other permissions are detailed in the documentation.

Repository creation policies cover whether a user can create a repository in the enterprise and, if so, what its visibility can be. Repository creators are automatically granted administrative permissions. Each option comes with its own set of considerations:

- If a "members can create repositories" policy is set at the enterprise level users are able to create repositories in organizations in the enterprise.
- More granular control is available by selecting "No policy, " which allows each organization to set its own policy.
- Selecting "Disabled" will prevent your users from creating repositories in the enterprise. Companies opting for this approach typically use an automated process to create repositories from standard configurations with the necessary associations to meet audit requirements. Some large enterprise customers prefer retroactive configurations applied post-create to ensure configuration standards.

Companies using Enterprise Managed Users may want to choose to enable the option to "Block the creation of user namespace repositories", because enterprise policies do not currently apply to repositories in personal namespaces. Doing so is not advised, however, if your company makes heavy use of forking or chooses to leverage those personal spaces as a "sandbox" for developers.

Copilot

Policies in this section govern use of Copilot, GitHub's AI pair programmer. Copilot is licensed separately from GitHub Enterprise Cloud. If your enterprise is using Copilot you can use the settings here to:

- Manage which organizations' users have access to the Copilot service
- Allow, or prevent, suggestions from the Copilot service that match public code

Actions

Policies in this section govern use of GitHub's automation and CI/CD tool, known as GitHub Actions.

Actions can be enabled on all, some, or none of the organizations within the enterprise. When enabled, administrators have the ability to define which specific actions are available to their workflows. Three options are available:

- "Allow all actions and reusable workflows" will let users run workflows containing any GitHub Action from the marketplace or defined in any public repository. This setting is not appropriate for most enterprises.
- "Allow enterprise actions and reusable workflows" may seem like the best choice for enterprises, but it often isn't. If you select this option ALL Actions must be defined in your enterprise. This means developers cannot consume Actions directly from the GitHub marketplace. You will need to define a process to clone or fork all desired

actions into your enterprise, train your teams to reference those internal actions, regularly check for updates, and bring them into your clones.
- "Allow enterprise, and select non-enterprise, actions and reusable workflows" is the sensible, manageable choice for most enterprises. Selecting this option presents an allow list administrators can use to individually authorize:
  - GitHub-created Actions
  - Actions from verified creators (authorship is verified by GitHub)
  - Specific third party actions and reusable workflows.

Enterprises will need to create and maintain their own process for requesting access to GitHub Actions when implementing the last option. Administrators will have the option to reference approved Actions in four ways:

- the simple owner/repo path to the Action's source code
- a specific branch of the Action, referenced as owner/repo@branch
- a specific tag/release of the Action, referenced as owner/repo@tag
- a specific commit SHA of the Action, referenced as owner/repo@SHA

Enterprises performing their own security reviews of third party Actions should use the owner/repo@SHA syntax specific to the version of the Action approved as commit SHAs are immutable while code referenced by branch names and tags can change.

If your organization will make use of fork-and-pull workflows you can define how Actions should respond to workflows initiated by pull requests into the parent repository. These settings are especially important if you will maintain open source repositories in your enterprise or if you make use of outside collaborators.

Projects

Policies in this section govern use of GitHub projects, the developer-centric, built-in project management capability of GitHub. Administrators can use this section to, optionally, disable use of projects in one or more organizations and govern visibility changes of projects.

Organizations

This section allows administrators to enable/disable dependency insights, which includes aggregated information about the use of OSS dependencies, their licenses, and any associated security vulnerabilities. This should remain enabled as part of a comprehensive strategy to understand OSS consumption, mitigate security concerns, and maintain license compliance.

Code security and analysis

This section includes policies governing use of GitHub's security features. Administrators can decide whether repository administrators should be allowed to

enable or disable Dependabot alerts. Most enterprises should set this to "not allowed," which will keep Dependabot enabled across all their code.

Users of GitHub Advanced Security will find additional, similar policy controls in this section. If you are using Advanced Security you'll likely want to keep all of these set to "not allowed," which will prevent individual repository owners from circumventing enterprise-wide security settings.

Supplemental Controls

While the policies section of the enterprise account covers a wide variety of enterprise needs, your specific requirements may necessitate the enforcement / addition of additional policy controls. The GitHub community has built a number of GitHub applications that extend our core platform to flexibility define, and enforce, unique policies. A few of the most common are listed below:

- Safe-settings: Safe-Settings is an event driven GitHub Application that prevents configuration drift within your organization. It is supplemental to the existing controls administrators enforce at the Repository, Organization and Enterprise Account levels.
- Policy-bot: Allows for the definition, and enforcement, of robust policies governing code review and approval of Pull Requests
- GHAS compliance: GHAS compliance is a policy as code, event driven GitHub Action that enables administrators to configure their Risk threshold for security issues reported by GitHub Code Scanning, Secret Scanning and Dependabot Security.
- Jira: Easily connect one or more GitHub Organizations to your Jira site and select specific repositories to bring your work together. Use the Smart Commits syntax to connect GitHub and Jira together.

Managing Billing

The enterprise account is the central point for all billing within GitHub Enterprise Cloud. This includes all organizations that are part of your enterprise (organizations are covered in depth in their own section, below).

Seat Licenses

GitHub Enterprise Cloud licenses are provisioned to your enterprise upon purchase and available for use. Users provisioned in your enterprise consume a license and can be added to any of your organizations. If you are using GitHub Advanced Security those licenses are purchased and managed the same way.

Consumption-based services

Use of services such as GitHub-hosted Actions runners, Packages, and Codespaces, are billed per unit consumed. Consumption across all of your enterprise's organizations is aggregated for invoicing. Customers who purchase GitHub Enterprise with a credit card are billed directly on a monthly basis.

Though charges are aggregated into a single invoice, a detailed, per-organization report of all consumption services are available. An enterprise-wide spending limit for each service can be set in your enterprise settings. In the near term, we plan to enable organization level spending limits for more fine grained control.

If you purchased GitHub Enterprise Cloud through a Microsoft Enterprise Agreement you can connect an Azure Subscription ID to your enterprise account and have all charges managed via your Azure invoicing. If you purchased GitHub directly on invoice, all consumption charges will be billed via invoice.

Some centralized tools teams may need to apply a charge back to individual teams operating within the environment and consuming resources. During migration, we recommend building a service catalog that maps repositories to projects and teams. This catalog could be kept in sync if repository creation is externalized. Otherwise, repositories should have their cost center ID as a repository topic. If there is no topic, that repository can not request to have consumption services enabled.

Insight into consumption services billing, including Advanced Security is provided at the Organization and Enterprise levels. In some cases, reports should be programmatically parsed for mapping aggregate consumption back to project teams.

Advanced Security

GitHub Advanced Security is licensed per seat for active committers to repositories using the features. Each license for GitHub Advanced Security specifies a maximum number of accounts, or seats, that can use these features. A committer is considered active if one of their commits has been pushed to the repository within the last 90 days, regardless of when it was originally authored.

Compliance Reporting

GitHub provides our latest SOC reports, 3rd party attestations, and self-reported business operations procedures as self-serve documents within the enterprise administration page.

Support

Enterprise administrators can open and manage their GitHub support tickets through the Enterprise landing page.

Organizations

Your enterprise will consist of one or more organizations. Organizations are shared accounts where your users collaborate across many projects at once, with sophisticated security and administrative features.

The structure you apply to your organizations can greatly facilitate collaboration and discovery while reducing administrative burden - or it can create unnecessary silos and add administrative overhead. As such, it is important to give careful consideration to organization structure ahead of time.

Organization Structure

Don't overly index your GitHub organizations to your current corporate structure.

When setting up your GitHub Enterprise instance, the immediate "least privilege" instinct may be to create an Organization for every project or department at your company. This might seem like a good way to manage permissions and reduce noise, but it's not the ideal strategy and should be avoided. It:

- Does not allow for fluid organizational structures. Your GitHub configuration should gracefully handle corporate reorganizations without having to rename organizations and handle all the resulting downstream work, such as re-pathing integrations and updating external links.
- Increases the team management burden and silos conversations. Teams (discussed later in this document) are currently an organizational construct. This means a team defined in one organization cannot be @ mentioned in another organization. If a team needs access to more than one org, the team must be created in each organization and mapped to the appropriate IdP group.
- Increases the burden of managing integrations. GitHub Apps are installed in organizations. Enterprises with many organizations must install and manage each of their integrations in each of their organizations. However, note that there is a limit of 100 GitHub App installations per Organization.

We want you to get started with an architecture that helps your team work together seamlessly, creatively, and transparently without bogging you down in unnecessary overhead. Instead of creating many Organizations and siloing users, we suggest using one or few Organizations for shared ownership of repositories and making use of Teams to segment users within those Organizations.

Below are three common models successfully adopted by GitHub Enterprise Cloud customers that effectively utilize as few organizations as prudent for their business:

- Single Organization
- Red / Green Organizations
- Portfolio Company Organizations

Model: Single Organization

In this model, a single organization is used for all (or the vast majority) of all repositories. Many small to medium-sized organizations (less than 5,000 developers) that largely operate as a single company effectively use this approach to manage their GitHub environment.

Most users of this model set their organization's base permissions to "none." This requires all users to be explicitly added to all repositories in order to view them and/or propose changes. By itself this approach creates silos, greatly limits visibility, and actively prevents Innersource. To prevent this, enterprises using this model use teams to ensure visibility. For example, an "all members" team can be created and added to repositories open to collaboration. Some have gone as far as to have this type of team automatically added to all repositories by default. An exception process exists to create repositories that must truly be kept on a need-to-know basis.

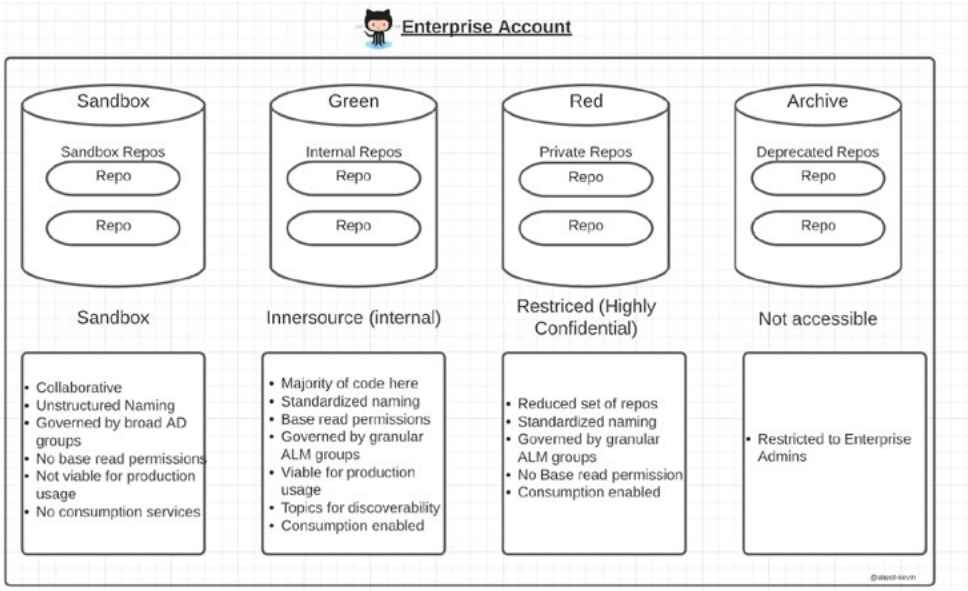
Exceptions

Slight variations on this model exist to handle extenuating circumstances. Some customers using this model maintain one or two "top secret" organizations for projects that must be kept completely separate from all other work, like projects for highly-sensitive customers.

Another variation on this model uses a separate organization to segment repositories managed by teams in particular locations. Enterprises taking this approach tend to have concerns around intellectual property laws.

For Organizations with 10s of thousands of developers, it may be practical to limit the number of developers in an organization to approximately 10,000. So a developer community of 70,000 may have 7 Green Organizations.

Model: Red / Green / Sandbox / Archive



The Red/Green model utilizes two primary organizations and a sandbox:

"Green" organization

The "green" organization serves as the primary collaboration space for developers operating within your enterprise account. About 90% of an enterprise's repositories will reside here. Setting the base permissions to "write" promotes Innersource and empowers users to propose changes to repositories. Effective use of branch protections (discussed later in this document) on repositories in the organization ensures users can only propose changes. Any change proposed will need to go through the defined code review and approval process to be accepted.

Higher-level permissions to repositories are granted through teams. Each team can use their team's repository page to view the repositories to which they have been granted additional permissions. With most activity happening in the "green" organization, the burden of managing teams and integrations is minimized.

"Red" organization

The "red" organization is for repositories that must be kept on a "need-to-know" basis. The default permissions of the organization are set to "none," which requires teams to be explicitly added to the repositories to read them or propose changes.

A defined process should exist for creating repositories and teams in this organization. Doing so prevents unnecessary, self-imposed constraints.

Optionally, two more organization may be added to this model:

"Sandbox" organization

A "sandbox" organization is a place in which any user can create a repository. This allows developers to experiment in a place more visible, and more collaborative, than personal repositories. Such an organization is especially important if you configure GitHub Enterprise Cloud to prevent developers from creating personal repositories.

A process for moving work from the sandbox into the green or red organizations should be defined to ensure experiments can transition into more formal management.

"Archive" organization

An "archive" organization to contain repositories no longer actively maintained. Any repository can be archived, making it read-only for all users. While these repositories can remain in their existing organizations, some enterprises prefer to transfer ownership of these archived repositories to a separate organization, leaving their other repositories for active work. Note that choosing to do this changes the "owner/name" syntax of the repos. This may make finding the repository in GitHub a bit more challenging if the user expects it to be in the original organization.

Model: Portfolio company (Merger, Acquisition, Divestitures)

There are a few scenarios in which the red/green model, described above, may not adequately match your enterprise's internal structures:

- Very large companies (10s of thousands of employees) divided into relatively static business units. Example: a global bank consisting of retail banking, personal investments, capital markets, and insurance divisions.
- "Portfolio" companies with operating units functioning largely independent of one another. Example: A media company consisting of: broadcast networks, production studios, streaming service, and interactive games companies. These companies may also experience regular mergers, acquisitions, and divestitures.

In a very large company we recommend mapping your GitHub Enterprise Cloud organization structure to your highest-level corporate division. These will generally be divisions one level below the CEO and be static in nature (reorganizations typically happen within the division, leaving the top level structure alone).

In a portfolio company organizations should be mapped to each business entity.

Organizations can be transferred from one enterprise to another, easing the burden of managing a merger or divestiture.

If possible, we recommend keeping a single organization per portfolio company or major business division. This is essentially multiple implementations of the single organization strategy described above. At most, we recommend implementing a minimal red/green structure for each major division. If you go down this path, consider implementing a single sandbox across all units and keeping archive repositories in the primary organization. This will help minimize the number of active organizations to manage.

Open Source

Open source projects should be kept in a dedicated organization, separate from the corporate org(s). Otherwise you can run into security boundary issues, particularly when you use outside collaborators. If you are using an Enterprise Managed User environment this may require having separate Enterprise accounts given the constraints on the former.

Repositories

Repositories are the core construct within which developers build software. Controls are available to manage work associated with the repository including issue tracking, managing proposed changes, and code reviews.

Prefer teams for permissions

Permissions can be granted to teams and individuals. When possible, Enterprises should grant permissions solely to teams, whose membership can be synced with IdP groups. Automation can be created, using webhooks and the GitHub API, to revert any individual user permissions granted. GitHub's open source safe-settings app implements a form of this, enforcing permissions defined in a configuration file.

Base permissions

The earlier section on organizational models advocated for defaulting repository permissions to "write" when possible. GitHub provides three different types of default repository permissions:

- "read" allows all organization members to see the repository, clone it, and participate in issues. If forking is allowed in your enterprise users can fork the repository.
- "write" allows all organization members the ability to push code to the repository. It

does not necessarily let members push to the default branch or approve their own changes. Properly configured branch protections, discussed below, control how members can modify the code.
- "admin" allows all organization members to administer the repository. This should not be used as a default.

Repository roles

Base permission levels for repositories are fairly coarse-grained. GitHub provides an expanded set of repository roles, which can be explicitly granted to individuals or teams. They are:

- Read
- Triage
- Write
- Maintain
- Admin

When planning your GitHub Enterprise Cloud implementation you may find a need for even more granular permissions than those applied to the default set of repository roles. If that happens, you can create custom roles for your organization and apply more fine-grained permissions.

Branch protections

Permissions to Git repositories are, by design of the standard, coarse. Without a mitigating control in place a user with "write" permissions to a repository can push commits to any branch, including default. To prevent this, teams should establish branch protection rules for their repositories.

A variety of protections are available to help teams map their desired processes to their branching strategy, and include:

- Requiring a Pull Request and code review for all commits merged. Most teams will enable this on their default / `main` branch they merge features into
- Requiring automated review(s) to pass. External processes such as GItHub Actions (CI/CD) and Advanced Security (code scanning) can create status checks on pull requests. You can define which checks must successfully pass before a pull request can be merged.
- Requiring pull request approvals from different teams, defined in a CODEOWNERS file, based on what file(s) were changed.
- Restricting the user(s) who can push to the branch. This can help teams prevent humans from pushing to branches managed by machine accounts as part of a branch-based release workflow. It can also be used to limit which users are allowed

to merge a pull request, if mandated by policy.
- Requiring signed commits. This feature requires users' commits to the repository by cryptographically signed, increasing confidence in their authorship.

Enterprises may wish to require a base level of branch permissions on each repository in an organization. This is possible through use of the open source safe-settings app (created by GitHub), or by using GitHub's webhooks and APIs. Safe-settings will let you define repository settings, such as branch protections, as code in a single repository. The app can apply those policies to all existing and newly-created repositories.

Those looking to enforce complex pull request approval workflows beyond that offered by required reviews and CODEOWNERS should consider implementing the open source palantir/policy-bot application.

Topics

Repository topics are a useful mechanism for grouping like repositories and promoting discovery. Use topics to tag repositories that are:

- all components of a larger system or process, i.e. "`payment processing`", "`mobile access`", or "`data modeling`",
- written the same language and/or framework, i.e. "`Spring Boot`", "`React`", or "`Rust`"
- maintained by the same team

Topics are mutable by design.

Teams

Teams to manage permissions

Earlier in this document we advocated for using teams, instead of organizations, to manage repository permissions and visibility, facilitate conversation, and reduce management overhead. Teams used for permission management should be synced with an IdP group. This allows existing IdP processes and audit controls to be relied upon for managing access to code. Onboarding, offboarding, and access changes are all managed by the IdP.

Teams to facilitate communication

Permissions aren't the only use for teams. GitHub teams can be used to engage project teams (not explicitly defined in your IdP) and/or topic-based teams in conversations:

`<Graphic of a PR conversation>"Hey @octocorp/react, I'm having trouble`

`figuring out why this component isn't updating along with the others. Could someone more familiar with this component give it a :eyes:? Thanks.`

They can also be automatically assigned as reviewers to code using CODEOWNERS.

`<Graphic of PR assignees, including a team added by Codeowners>`

`<Graphic of a simple CODEOWNERS file, showing the team added in the above graphic>`

If your enterprise does not have a simple, quick way to create and manage these groups in your IdP, organization members should be able to create and manage their own ad-hoc teams.

Team pages

Each team has its own page within an organization, which takes the form `https://github.com/orgs/<orgName>/teams/<teamName>`. Alternatively, you can navigate to a team's page from your organization's main page (`https://github.com/orgs/<orgName>`), clicking on teams, then clicking on the desired team.

Team Repositories

Enterprises commonly desire to easily see which teams maintain which projects. And members within teams appreciate a canonical list of the code their team maintains. To meet both of these needs a list of all repositories to which a team has been granted explicit permissions (beyond the organization's default) is available on each team's page in GitHub.

The team repositories page can be accessed by clicking on "Repositories" in the top bar, or directly accessing it at `github.com/orgs/<myOrg>/teams/<myTeam>/repositories`.

Users

Additional considerations for Enterprise Managed Users

The earlier section on enterprise account configuration detailed the differences between standard and enterprise managed users and how to choose between the two. If you elect Enterprise Managed Users a few additional configuration options are present.

Personal namespaces

By default, users can create repositories in their namespace. The user who creates the repository is, by default, an administrator and can invite other users to collaborate on the repository. Team-based permissions are not available as teams are owned by organizations, and personal namespaces are outside of any organization.

Some enterprises prefer to allow individual developers the freedom to experiment in their personal namespaces. Those who do should have a defined process for promoting experiments to organization-owned repositories.

Other enterprises prefer to give developers the ability to create experimental repositories within an organization, often a "sandbox". Keeping such repositories in an organization lets them apply organization-wide policies to those experiments and use teams to manage access. This behavior can be enforced by blocking the creation of user namespace repositories

Restricted users for partners / consultants

Managing access for partners and consultants presents a unique challenge. Such users often need access to several repositories, and to participate in team conversations. However, adding them as full members of the organization would grant them the default access given to all other organization members, which may be too broad.

For this reason, GitHub provides a "restricted' user type. Restricted users in an EMU environment operate as standard users, but they do not have access to internal repositories.

Platform security (secure your environment)

Authentication & Authorization

Establishing controls

- Push Protections
  - When you enable push protection, secret scanning proactively checks pushes for high-confidence secrets (those identified with a low false positive rate). Secret scanning lists any secrets it detects so the author can review the secrets and remove them or, if needed, allow those secrets to be pushed.

- In addition to push protections for Secret Scanning, GitHub is developing (or has developed) additional Organization controls to limit pushes that are too large, containing invalid extensions (.mov, etc), non compliant commit messaging and file path length limits.

- Code In/Exfiltration
  - Some organizations apply broad network controls to limit developer access to github.com to primitively address the risk of data exfiltration. However, GitHub UI traffic can be managed through an enterprise session decorator indicating an active EMU session that inherently prevents write access to public GitHub.com. If the decorator exists, traffic to GitHub is allowed because constraints are automatically applied. If the decorator does not exist, traffic is further evaluated. This doesn't satisfy all aspects of malicious exfiltration but it does provide another layer of control to prevent inadvertent publishing to public GitHub. Consider monitoring traffic to GitHub at the proxy or network gateway for this decorator.
  - Similar to Stack Overflow, Medium or miscellaneous technical posts, with unfettered access to GitHub.com, developers may have access to code with malicious intent. While access to open source code has a measurably positive impact on the quality and velocity of software delivery Organizations should manage its consumption thoughtfully. There are layers of controls within most organizations that limit the blast radius and efficacy of that code including:
    - Segmentation within sensitive environments
      - Developers may execute malicious code, but if that code doesn't have access to sensitive data or environments, it's blast radius is reduced. Typically, there are processes in place for managing code escalation between environments that should catch vulnerabilities prior to exploitation.
    - Automated SAST and SCA scans prior to deployment
      - In the code promotion process from development, through QA and into production automated code scans should be enforced to provide feedback to the developer and their team as to whether the code being introduced is secure. GitHub provides Advanced Security features for this purpose, deeply integrated with the developer workflow for in context remediation.
    - Private registry enforcement
      - Packages pulled from GitHub and promoted through initial stages of development are typically unavailable to build agents who only have access to Organization approved private registries, excluding public registries. Packages shunted into an application that are unlisted on the application manifest are subject to standard SAST scans.
    - Network in/egress controls
      - Code that attempts to, for example, reach out to a malicious URI would be prevented at the network level through standard network egress controls. All inbound traffic from GitHub comes from known, cryptographically verified locations.

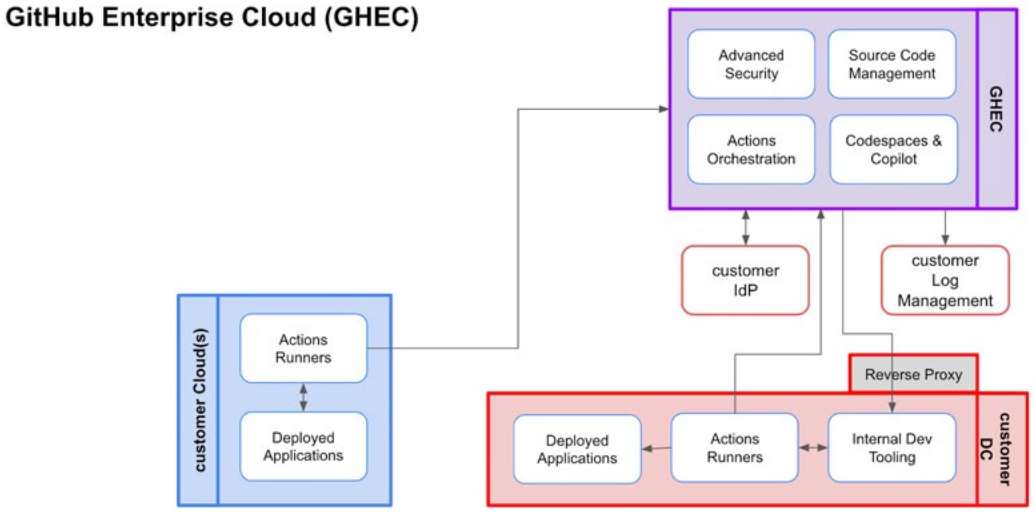
Personal Access Tokens (PAT) & SSH Keys

- Classic
  - Personal access tokens are an alternative to using passwords for authentication to GitHub when using the GitHub API or the command line. Personal access tokens are intended to access GitHub resources on behalf of yourself. To access resources on behalf of an organization, or for long-lived integrations, you should use a GitHub App.
- V2
  - Modern personal access tokens provide more fine grained access to Organization resources. In contrast to most classic tokens, they also have finite lifespans - defined during credential creation.
- MFA
  - For supported IdPs MFA is typically handled within that environment. However, GitHub also provides native MFA for non-EMU organizations.

Integrations with Existing Tooling

Enterprises are dutifully hesitant to create a porous network boundary for their self hosted resources. As such, integrating a SaaS service such as the GitHub platform requires a thoughtful, layered approach. The following integration architecture and design patterns ensure secure connectivity between GitHub Enterprise Cloud and your self hosted resources.

Common integration architecture



Webhooks

Webhooks allow integrations to subscribe to events on GitHub and receive notification when those events occur. Integrations create webhooks, select the events for which they desire notification, provide a publicly-routable HTTPS endpoint to which GitHub can deliver a JSON payload, and (optionally) a secret value GitHub can use to cryptographically sign the payload.

Webhook payloads are JSON delivered in an HTTPS POST.

Delivering webhooks to on premises resources

As part of your deployment of GitHub Enterprise Cloud you may need to have webhooks send notifications to applications hosted behind your firewall. The two most common needs are sending notifications to locally-deployed CI/CD solutions, like Jenkins, and to locally-deployed project management tooling, like Jira Server.

With proper network configuration it is possible for GitHub Enterprise Cloud to notify these systems. This is achieved through implementing a form of reverse proxy, which includes:

- A publicly-routable FQDN to which GitHub can send the HTTPS post
- URL rewrite and port forwarding to the appropriate application

This pattern can be implemented using off-the shelf components from vendors like ngrok, configuration of existing web servers like NGINX, or a commercial product such as a WAF or API gateway.

Securing webhook deliveries to on premises resources

When implementing a reverse proxy to handle webhook delivery GitHub recommends the following practices:

- Only allow inbound HTTPS traffic on port 443 to your FQDN
- Only allow traffic from the IP ranges advertised in the `hooks` section of https://api.github.com/meta
- Terminate SSL at your FQDN and inspect the JSON payload for the presence of well-formatted JSON
- Ensure the solution(s) you are integrating with support, and use, webhook signatures

HTTPS requests from GitHub.com containing information about an event that has happened on the platform. They serve as the preferred mode of integration for service to service communication.

Copilot endpoints

https://github.com/login/device/code

https://github.com/login/oauth/access_token

https://api.github.com/user

https://api.github.com/copilot_internal/v2/token

https://api.github.com/copilot_internal/notification

https://default.exp-tas.com

https://copilot-telemetry.githubusercontent.com/telemetry

https://copilot-proxy.githubusercontent.com

API access

IP allow lists

Integrations with GitHub may need to call GitHub's APIs in order to properly function. If you have configured an IP allow list for your organization the IP ranges of any integrations installed must be included in the allow list. When setting up the IP allow list you can optionally allow installed integrations to update it. If the author of the integration provides an allow list for their application it will automatically be added to your allow list. If not, the allow list will need to be managed manually.

IP allow lists and Conditional Access Policies for EMUs

If you are using Enterprise Managed Users with OIDC SSO GitHub will automatically use your IdP's conditional access policy (CAP) IP conditions to validate user interactions with GitHub. You will need to ensure the IP ranges of all applications and integrations are added to your IdP's CAP.

IP allow lists, CAP, and Actions runner access

GitHub provides hosted Actions runners you can use to perform your builds, deployments, static analysis scans, and any other desired automations. If you are using GitHub-hosted runners with 4 or more vCPUs you can receive a static IP address for your runners. This IP address is unique to your enterprise and can be added to your IP allow list or IdP CAP to grant the runners access to your code.

Polling

If you are writing an integration or automation against the GitHub API you should avoid API polling. To ensure the performance and reliability of our system for all GitHub enforces a hard rate limit on API activity for individuals and a higher limit for applications. Regular polling of our API, especially in larger organizations, will hit the rate limit.

Rather than polling our API on a scheduled basis, you should instead use webhooks to be notified of events on GitHub. The integration will receive any webhook events to which it is subscribed and can then make an authenticated request to GitHub should any action need to be taken.

Network Configuration

- To connect with GitHub.com from an isolated workspace, consider allowing scoped communication with the service based on known URIs. URIs are available based on request and per service.
- GitHub strives to deliver a modified session header that includes the enterprise identifier for EMU accounts. This identifier, attached to all requests, would provide key data for network filtering to permit write authorization exclusively to the enterprise account and read to all of GitHub.

Migrations

Changing to a new system can seem like a daunting task. Thankfully, GitHub provides a number of helpful tools to ease the technical aspects of migration and services to help your organization establish a plan for successful migration with minimal impact to teams.

Migrating step by step

Version control, CI/CD, and security tools and processes can get quite complex. Given the complexity, most enterprises focus first on migrating their source code to GitHub Enterprise Cloud, while supporting existing tooling such as planning and tracking and CI/CD. Moving all code from disparate version control systems into GitHub lets developers quickly start realizing the benefits of GitHub:

- Code is more easily searched, shared, and collaborated on.
- Open Source dependencies are immediately monitored by Dependabot.
- Code can be scanned and monitored for secrets by GitHub Advanced Security.
- Cloud-hosted dev environments reduce the on-boarding and context switching costs associated with switching between projects

This migration is usually performed team-by-team or business unit by business unit. Once a team's code is in GitHub, and integrated

with existing tooling and processes, those teams can begin migrating their:

- Security scanning process to GitHub Advanced Security's code scanning
- CI/CD jobs to Actions

A source code migration creates the opportunity for a new AppSec paradigm, shifting security left into the developer workflow. Lastly, CI/CD and aspirationally planning and tracking.

Consider enabling **GitHub Copilot** early in the code migration process to deliver a quick and incredibly impactful win for your organization. Copilot not only makes developers more productive, but it measurably improves their satisfaction and reduces the friction of adopting new tools like Actions by creating templates automatically.

Migration tooling

GitHub provides tooling to help you easily migrate your repositories and CI/CD pipelines into GitHub.

The GitHub Enterprise Importer (GEI) is a highly customizable API-first migration offering designed to help you move your enterprise to GitHub Enterprise Cloud. The GEI-CLI wraps the GEI APIs as a cross-platform console application to simplify customizing your migration experience. Most enterprise customers who are using the CLI, often abstract the CLI for their developer communities by building a dashboard on top of the CLI. That abstraction can contain a series of validations for the migration including checks on repo/file size and lfs usage. Contributions can be mapped back to original committers during the migration process.

The GitHub Actions Importer helps facilitate the migration of Azure DevOps, CircleCI, GitLab CI, Jenkins, and Travis CI pipelines to GitHub Actions.

Summary

GitHub, as a wholly owned subsidiary of Microsoft strives to continuously provide an experience developers know and love so that they can deliver experiences that delight your customers. GitHub Enterprise Cloud is an enterprise market leading, developer first, software delivery platform. In a world consumed by software, GitHub is the advantage.

Learn more at github.com/enterprise