GitHub Certifications - Study Guide

Last Updated: 5/28/2025

This document is a compilation of study guides I've created throughout my preparation for the GitHub Certifications. It brings together key concepts, summaries, and practice insights that helped me understand the material more effectively. I'm sharing it with the team in hopes that it serves as a helpful reference and supports your own study journey. Please feel free to build on it, adapt it to your needs, and share back any improvements! Good Luck!

GitHub Foundations

What is the GitHub Foundations Certification?

Entry level certification teaching:

- Git Version Control Basics
- Working with GitHub Repositories
- Collaboration Features
- Modern Development
- Project Management
- GitHub Privacy, Security, and Administration
- GitHub Community and Open Source

Common Git Terms:

- *Repository:* Logical container holding the codebase
- *Commit:* A change of data in the local repository
- *Tree:* The entire history of a repository
- *Remote:* A version of your project hosted elsewhere, used for exchanging commits
- *Branches:* Divergent paths of development, allowing isolated changes
    - **Main**: Most common name for the default branch
- *Clone:* Creates a complete local copy of a repository, including its history
- *Checkout:* Switches between different branches or commits in your repository
- *Pull:* Downloads changes from a remote repository and merges them into your branch
- *Push:* Uploads your local repository changes to a remote repository
- *Fetch:* Downloads data from a remote repository without integrating it into your work
- *Reset:* Undoes local changes, with options to unstage or revert commits
- *Merge:* Combines multiple commit histories into one
- *Staging files:* Prepares and organizes changes for a commit.
    - Commit: Saves your changes as a snapshot in the local repository
    - Add: Adds changes to the staging area for the next commit

Git Crash Course:

- *Cloning:* There are three ways to clone: HTTPS, SSH, GitHub CLI
- *Logging:* git log will show recent git commits to the git tree
- There is a hidden folder called ".git" which tells you that your project is a git repository
    - If you want to create a git repository in a new folder you can just use "git init"

Git Commits:

- A Git Commit represents an incremental change to a codebase represented with a git tree (graph) at a specific time
- Each commit has:
    - A Commit SHA hash that acts as an ID, you can use this commit sha to checkout specific commits.
    - It has the Author information, the name and email of the person who made the commit.
    - The commit message which is a description of what changes the commit contains.
    - The timestamp which has the date and time when the commit was made.
    - Parent Commit Hash: Which is the sha hash of the commit(s) this commit is based on.
    - Snapshot of the content, snapshot of the project at the time of the commit. Not the actual files, but references to them.
- A git commit contains additions, modifications, and deletions of files. Additions and deletions of file contents. It doesn't contain the whole files themselves, only changes. This greatly reduces the file size.

Git Branches:

- A divergence of the state of the repository.
- Branches can be thought of as being copies of a point in time that have been modified to be different.

Git Remote:

- A git remote represents the reference to a remote location where a copy of the repository is hosted.

SSH Keys vs PAT:

- PAT's are useful in a variety of situations. I believe the primary use case is when you need to access a repo as 'you' without risking exposing your private SSH key.
- For example if you have some script that's accessing GitHub API and only need read access, generate a PAT with read only permissions for that purpose. If that were ever exposed, deletion doesn't interfere with your day to day work and the exposure wouldn't include write access.
- You can consider PAT's like a 'guest' key to your house where you define what they can access and can easily revoke that permission.

- SSH keys work by showing you can encrypt a secret that can be decrypted by your public key. So your key never leaves your machine.
- Unlike a PAT, which you transmit with each request.
- Which makes stealing a PAT a lot easier than stealing a key. (Incidentally, you can use a ssh-agent to store your keys for you.).
- As to the granularity, deploy keys and group keys exist for a reason. But you have indeed more control with PAT's there. In my view the SSH keys' better security profile outweighs the benefits of granularity control of using PATs.

GitHub APIs:

- There is a Rest API and GraphQL API
- Pros and cons to each, for the Rest API you need to use PAT
- GraphQL is built in the web

GitHub SDKs:

- Octokit - the official SDKs to programmatically interact with the Git Rest API
- GitHub officially maintains SDKs for Javascript/Typescript, C#/.Net, Ruby, Terraform provider,

GitHub Desktop:

- Standalone application to interact with GitHub repos without the browser or via code.
- Common Git and GitHub Operations can be performed via the GUI for an easy to use experience.
- Manages local copies of repositories

GitHub Mobile:

- A mobile application you can install on phones to perform read only or basic GitHub repo management tasks
- Manage, **triage,** and clear notifications
- Read, review, and collaborate on issues and pull requests.
- Edit files in pull requests
- Search for, browse and interact with users, repos and orgs
- Receive a push notification when someone mentions your username
- Search through code in a specific repo
- Secure your GitHub.com account with two-factor auth

GitHub Mobile Notifications:

- You can set what push notifications you want to receive:
    - Direct Mentions,
    - Review Requested
    - Assigned
    - Deployment Review
    - PR Review

- Workflow Runs

Types of GitHub Accounts:

- Personal: These are individual accounts with a username and profile, they can own resources like repositories and projects, and actions taken are attributed to the personal account.
    - With a GitHub personal account you can have a public profile and host your own public website, repo urls for personal accounts include usernames
- Organizational: Shared accounts where multiple people collaborate on projects. They can also own resources like repositories but are managed through individual personal accounts. Organizations can offer different roles with varying levels of access and come with security features.
    - GitHub organizations can also have their own public profile page.
- Enterprise Accounts: Part of GitHub Enterprise Cloud and Server, these accounts allow for central management of multiple organizations. They're geared towards larger setups needing centralized policy and billing management or internal GitHub repos.

GitHub Free vs Pro:

- ***GitHub Free:*** GitHub Community Support, Dependabot Alerts, Deployment protection rules for public repositories, Two factor authentication enforcement, 500 MB GitHub Packages Storage, 120 GitHub Codespaces core hours per month, 15 GB GitHub Codespaces storage per month. GitHub Actions features: 2,000 minutes per month, deployment protection rules for public repositories.
- **GitHub Pro:** Everything from GitHub Free AND GitHub Support via email, 3,000 GitHub Actions minutes per month, 2 GB GitHub Packages Storage, 180 GitHub Codespaces core hours per month, 20 GB GitHub Codespace storage per month, Advanced tools and insights in private repositories: Required pull request reviewers, multiple pull request reviewers, protected branches, code owners, auto-linked references, GitHub Pages, Wikis, Repository insights graphs

GitHub Organizations Plans:

- GitHub Organizations has three plans: Free, Teams, and Enterprise
- **Free:** Everything from GitHub Free and… Team access controls for managing groups, 2,000 GitHub Actions minutes per month, 500 MB GitHub Packages storage
- **Teams:** Everything from GitHub Organizations Free and… GitHub Support via email, 3,000 GitHub Actions minutes per month, 2 GB GitHub Packages storage, Advanced tools and insights in private repositories, Required pull request reviewers, Multiple pull request reviewers, *Draft pull requests*, Team pull request reviewers, Protected branches, *Code owners*, Scheduled reminders, GitHub Pages, Wikis, Repository insights graphs, The option to enable or disable Github Codespaces

GitHub Enterprises Deployment Options:

- GitHub Enterprise Cloud: When you want to have a hosted version of enterprise edition on GitHub.com.
- GitHub Enterprise Server: When you want to have a self hosted version of GitHub.
- *Both GitHub Enterprise Cloud and Server both receive:*
    - Everything from GitHub Organizations Team
    - GitHub Enterpise Support
    - Additional security, compliance and deployment controls
    - Authentication with *SAML single sign-on*
    - Access provisioning with *SAML or SCIM*
    - Deployment protection rules with GitHub Actions for private or internal repositories
    - GitHub Connect
    - Option to purchase GitHub Advanced Security
- *GitHub Enterprise Cloud Specific Features:*
    - 50,000 GitHub Actions minutes per month
    - 50 GB GitHub Packages storage
    - A service level agreement for 99.9% monthly uptime
    - Option to centrally manage policy and billing for multiple GitHub.com organizations with an enterprise account
    - Option to provision and manage the user accounts for your developers, by using Enterprise Managed Users

Markdown:

- A Markup language is a way of formatting and presenting text data in a different format.
- A common use-case for a markup languageis to present data in HTML.
- Markdown is a markup language that provides shorthand syntax to format information into HTML. Markdown is popular due to its easy syntax, and being readable in its raw format. File extensions are *.md
- Slash commands: On GitHub.com provides convenience features such as formatting markdown. Makes your life easier bu automating some markdown.

GitHub User Profile:

- For personal accounts, you can have a public GitHub User profile to showcase yourself as a developer
- Contributions Graph: Commits, PRs, Issues, and Code Reviews

README Files:

README is always in the project's root

General Repo Stuff:

- *Code:* The main tab where the repository's source code, files, and folders are located
- *Issues:* Tracks problems or ideas for the project, allowing collaboration and discussion.
- *Pull Requests:* Used for managing contributions from other users, enabling code review and discussion before merging change.

- *Actions:* Manages continuous integration and continuous deployment (CI/CD) workflows.
- *Projects:* A board for organizing and prioritizing work, similar to kanban or task management boards.
- *Wiki:* A space for the project's documentation
- *Security:* Features security-related resources, including security policies and vulnerability reports.
- Insights: Provides statistics and insights on the project's activity and contributions
- Settings: Contains repository settings, including access controls, webhooks, and other configurations
- Repo names are unique based on the scope of the owner and a repo can be either public or private
- When you're creating a repo you can quickly add a Readme file, .gitignore file, License file.
- You can change the repo if you need to
- You can change the base branch (default branch)
- You can opt-in-and-out of some features for your Github Repo like Wikis, Issues
- Danger Zone: Contains actions you need to think twice about because they cannot be undone.
- Branch protection rules: Strict workflow rules like disallowing anyone pushing to main. You can disable all protections temporarily if needed for quick fixes.
- You can easily find repos you've starred by going to github.com/stars
- Watching a repo allows you to stay informed about activities occurring within a repo.
- GitHub Releases allows you to create releases with specific release notes and linked assets such as zip source or binaries for specific platforms.

GitHub Packages:

- A platform for hosting and managing packages including containers and other dependencies
- Supported package registry: Javascript (npm), Ruby (gems), Java (maven and gradle), .Net, Docker Images

GitHub Issues:

- *Issues:* Tracking tasks, bugs, enhancements and other actionable items. Often linked to code changes, can be linked to PRs
- *Discussions:* Facilitating conversations and Q&As about a wide range of topics related to the project. Categorized by topics. Can be "converted" to issue. Really just creates an issue with a soft link to the issue. Not directly linked to code changes.
- *Pull Requests:* Proposing, reviewing and merging code changes into the codebase. Directly linked to code changes. Can be linked to issues.
- You can close an issue from a PR by doing stuff like "closes #10" or "resolve #4", just remember it's different variations of close, fix, resolve
- You can filter issues for open issues and prs, your issues, everything assigned to you, everything mentioning you

- Filters: Filters are the same for issue and PRs. Some options are Open Issues and Pull Requests, Your issues, Your pull requests, Everything assigned to you, Everything you're mentioned in
- You're also able to filter based on author, label, projects, milestones, assignee, and sort by newest, oldest, most commented, least commented, recently updated, least recently updated, best match
    - Example: "is:issue no:assignee label:activesupport comments:0 Caches"
- Issue Templates are markdown templates that are preloaded for new Issues. They help ensure users creating issues provide all the relevant and expected information.
- You can create multiple issues templates to better improve the context of issues
- On the exam they may ask what are the three default issues templates: Bug Report, Feature Request, Report a security vulnerability
- Issue templates are stored as markdown files in .github/ISSUE_TEMPLATES folder in your repo, GitHub has a wizard GUI to easily create Issue Templates
- Issue Forms: The evolution of Issues Templates. You use a YAML Formatted file to create issue forms for stricter entry of issue information. Beta feature.
- You can pin up to 3 issues

GitHub Pull Requests:

- A Pull Request (PR) is a formal process to put forth changes that can be manually or automatically reviewed before it's accepted into your base (main) branch.
- Benefits:
    - *Collaborative Review:* Enhances code quality through team discussions and peer feedback.
    - *Change Tracking:* Provides a record of code changes and related discussions.
    - *Automated Testing:* Enables integration with tools for automated checks and tests.
    - *Controlled Integration:* Manages safe and reviewed merging of code changes.
    - *Open Source Friendly:* Simplifies contributions and collaboration in open-source projects.
- A PR is not a feature of git but a workflow. Services like GitHub can automate the PR workflow.
- Base is who we are going to merge into, Compare/Head is the changes to pull in
- You can also compare across forks, this is how a fork stays up to date or how forks can suggest you to accept their changes
- Draft PRs allows you to open a PR but mark it as a work in progress. Cannot be merged. Code owners are not automatically requested to review draft PRs.
- You can link issues to a PR so that the state of the PR will automatically close the Issue, the PR must be on the default branch.
- When merging a PR there are a few options, Create a merge commit - all commits will be added. Squash and merge - 1 commit will be added. Rebase and merge - 1 commit will be added and rebased. Usually have squashed and merged.
- Required reviewers is a way to explicitly say these people have to review and approve the code.

- Pull Request Templates are in *.github/pull_request_template.md*

GitHub Discussions:

- Community communication tool for your public or private GitHub repos.
    - Thread conversations
    - Categorization
    - Community Interaction
    - Markdown Support
    - Pin Discussions
    - Conversation Polls
    - Discussion Voting
    - Conversion to Issues
    - Notifications
    - Searchable and Linkable
    - GitHub Integration
- You have to turn on discussions under your GitHub Repo Features
- A discussions tab will appear in your github repo. You'll have some default categories available.
- You can mark answers you think are correct in GitHub discussions
- You can convert a discussion to an issue

Notifications:

- There is a filter syntax that looks the same as every other filter in GH.
- Watching is for people, repos, or organizations. Notification Subscriptions are for specific PRs or Issues.

Codeowners File:

- CODEOWNERS GitHub repo specific file to define individuals or teams that are responsible for specific code in a repository. Uses similar syntax to .gitignore
- When a PR is open that modifies any files matching a pattern in the CODEOWNERS file, GitHub automatically requests a review from the specified code owners.
- The codeowner files goes in either the project root, .github, or docs directory

Gists:

- Gists provide a simple way to share code snippets with others. Every gist is a Git repository, which means that it can be forked and cloned. gist.github.com
- Gists can be public or secret
- Secret Gists are not private, you can share URL with friends. Secret Gists can be changed to public, Public Gists cannot be changed to be private
- You can pin Gists to your profile for others to easily find, Gists allow other users to comment. Gists allow you to easily navigate revisions
- Every Gist is a git repo, which means it can be forked and cloned.
- Gists can also be markdown.

GitHub Wiki Pages:

- A Wiki is a collaborative way to quickly create documentation with multiple nested pages, A wiki would serve as a knowledge base for your repo or organization
- You have to turn on Wiki in settings
- GitHub Wikis can be used for free in public
- When you create a page you can use various markup languages

GitHub Pages:

- GitHub Pages allows you to directly host a static website via a GitHub repository. You need to create a public repo with the repo name as <username>.github.io

GitHub Actions:

- CI/CD Pipeline directly integrated with your GitHub repository.
- GitHub Actions allow you to automate:
    - Running test suites
    - Building images
    - Compiling static sites
    - Deploying code to servers and more…
- GitHub Actions has templates you can use to get started.
- GitHub Actions files are defined as YAML files located in the .github/workflow folder in your repo.
- You can have multiple workflows in a repo triggered by different runs
- There are 35+ event triggers for Github Actions
- The **on** attribute specifies the event trigger to be used: Ex: on: [push]
- Examples of common GitHub Actions that could be triggered:
    - Pushes: Trigger an action on any push to the repository.
    - Pull Requests: Run actions when pull requests are opened, updated, or merged
    - Issues: Execute actions based on issue activities, like creation or labeling
    - Releases: Automate workflows when a new release is published.
    - Scheduled Events: Schedule actions to run at specific times
    - Manual Triggers: Allow manual triggering of actions through the GitHub UI

GitHub Copilot:

- AI Developer tool that can be used with multiple IDEs via an extension
- Copilot Individuals vs Copilot Business

GitHub Codespaces:

- Codespaces is a cloud developer environment integrated with your GitHub Repo. An environment is known as a codespace.
- A codespace runs in Ubuntu Linux docker container, on a virtual machine, hosted and managed by GitHub.

- GitHub.dev is free whereas Codespaces has a monthly quota. Codespaces has storage and core hours per month as well as machine type changes costs.
- Github.dev Editor is a VS Code browser that instantly loads that has no attached compute. It instantly launches and has no devcontainer.json where as Codespaces does and has a dedicated attached VM.

Open Source:

- Source code made freely available for possible modification and redistribution
- Open source benefits include:
    - Encourages global collaboration
    - Speeds up innovation
    - Offers adaptability and customization
    - Reduces software costs
    - Enhances learning for developers
    - Typically high quality and reliable
    - Provides transparency for trust and security
- Open source software often has free-community versions which makes it easy for personal developers or small organizations to quickly adopt technology.
- The OSI (Open Source Initiative) is a non profit org which is the steward of the Open Source Definition, the set of rules that define open source software
- The OSI maintains a list of open source licensing documents for all different use-cases which are readily available for project owners to adopt.

EMU (Enterprise Managed Users):

- Allows you to manage the lifecycle and authentication of your users on GitHub.com from an external identity management system or IdP
- For Partner IdP you can use Microsoft Entra ID, Okta, and PingFederate they all support SAML, only Microsoft Entra ID supports OCID, and they all support SCIM

Quick tips:

- If you type “.” in any GitHub repository it opens up github.dev which allows you to have a in browser IDE.
- CMD + K for the command palette works well

GitHub Copilot

Responsible AI with Copilot:

- To mitigate risks of AI systems making decisions that are difficult to interpret and can result in unintended and harmful outcomes, such as biased decision making or privacy violations it is important to implement robust governance frameworks, ensure transparency in AI processes and incorporate human oversight.
- Responsible AI is an approach to developing, assessing and deploying AI systems in a safe, trustworthy, ethical way.

Microsoft and GitHub's six principles of responsible AI:

- *Fairness:* AI systems should treat all people fairly
    - Training AI models on diverse and balanced data can help reduce biases, ultimately promoting fairness.
- *Reliability and Safety:* AI systems should perform reliably & safely
    - Safety in AI refers to minimizing unintended harm, including physical, emotional and financial harm to individuals and societies.
    - Reliability means that AI systems perform consistently as intended without unwanted variability of errors.
- *Privacy and Security:* AI systems should be secure and respect privacy
    - Getting users' permission before collecting their data.
    - Collecting only the data needed for the AI to work.
    - Anonymizing personal data.
    - Encrypt sensitive data both during transfer and when stored: HSMs, Secure vaults, Envelope encryption, Control who can access keys & models
- *Inclusiveness:* AI systems should empower everyone and engage people
    - Ensuring that AI systems are fair, accessible, and empowering everyone.
    - AI can't exclude diverse groups, geographies, communities, handicapped individuals.
    - Examples: Facial recognition that works across skin tones, interfaces that support screen readers for visibly impaired, language translation for small regional dialects, diverse perspectives when designing systems
- *Transparency:* AI systems should be understandable and interpretable
    - AI creators should be able to explain how their systems operate clearly, justify the design choices, be honest about the capabilities and limitations and enable auditability with logging and reporting
- *Accountability:* People should be accountable for AI Systems
    - AI Creators need to continuously monitor system performance and mitigate risks, AI systems must be accountable to people and companies

Introduction to GitHub Copilot:

- In the three years since its launch, developers have experienced the following benefits:
    - 46% of new code now written by AI
    - 55% faster overall developer productivity
    - 74% of developers feeling more focused on satisfying work
- GitHub Copilot is powered by the OpenAI Codex system, it is more powerful because it was trained on a dataset that included a larger concentration of public source code.
- Copilot is available as an extension for VS Code, Visual Studio, Vim, and JetBrains IDEs.
- Copilot Chat: brings a chatgpt like chat interface to the editor, focuses on developer scenarios and natively integrates with VS Code.
- Copilot for Pull Requests: AI Powered tags in PR descriptions through a GitHub app that organization admins and individual repository owners can install.
- Copilot for CLI: Can compose commands and loops and it can throw obscure find flags to satisfy your query.
- **Subscriptions:** GitHub Copilot is available to GitHub personal accounts with GitHub Copilot Free and GitHub Copilot Pro. It's available for organization accounts with GitHub Copilot Business, for enterprise accounts with GitHub Copilot Enterprise.
    - **GitHub Copilot Free:** Allows individual developers to use GitHub Copilot at no cost. Free tier includes 2,000 code completions per month, 50 chat requests per month, and access to both GPT-4o and Claude 3.5 Sonnet models,
    - **GitHub Copilot Business:** Allows you to control who can use Copilot in your company. After you give access to an organization, its admins can give access to individuals and teams. Copilot Business is focused on making organizations more productive, secure, and fulfilled:
        - Code completions
        - Chat in IDE and mobile
        - Filter for security vulnerabilities
        - Code referencing
        - Filter for public code
        - IP indemnity
        - Enterprise-grade security, safety and privacy
    - **GitHub Copilot Enterprise:** Available for organizations through GitHub Enterprise Cloud. Includes everything in Copilot business plus a layer of personalization for organizations. It provides integration into GitHub as a chat interface, so developers can converse about their codebase. GitHub Copilot Enterprise can index an organizations's codebase for a deeper understanding and for suggestions that are more tailored. This plan enables your teams to:
        - Quickly get up to speed on your codebase
        - Search through and build docs
        - Get suggestions based on internal and private code.
        - Quickly review PRs

Interacting with Copilot:

- **Inline Suggestions:** Most immediate form of assistance in Copilot. As you type, Copilot analyzes your code and context to offer real-time code completions. This feature predicts what you might want to write next and displays suggestions in a subtle, unobtrusive way.
- **Command Palette:** Open the command palette with Cmd+Shift+P. Enter Copilot to see commands. Select actions like explain this or generate unit tests to get assistance.
- **Copilot Chat:** Interactive feature that enables you to communicate with Copilot by using natural language.
- **Inline Chat:** Enables context-specific conversations with Copilot directly within your editor. You can use this feature to request code modifications or explanations without switching contexts. /explain /suggest /tests /comment
- Copilot learns from context, keeping your code well structured and commented helps Copilot provide more accurate and relevant assistance. The more you interact with Copilot the better it becomes at understanding your coding style and preferences.
- Example on how to get Copilot to introduce the project to me:

  @workspace Please briefly explain the structure of this project. What should I do to run it?
- You can use the run command on terminal suggestions and it'll run it for you
- *This is a really good issue to run through when creating a copilot demo*

Prompt Engineering:

- The core rules for the basis of creating effective prompts:
  - *Single:* Always focus your prompt on a single, well-defined task or question. This clarity is crucial for eliciting accurate and useful responses from Copilot.
  - *Specific:* Ensure that your instructions are explicit and detailed. Specificity leads to more applicable and precise code suggestions.
  - *Short:* While being specific, keep prompts concise and to the point. This balance ensures clarity without overloading Copilot or complicating the interaction.
  - *Surround:* Utilize descriptive filenames and keep related files open. This provides Copilot with rich context, leading to more tailored code suggestions.
  - Inbound Flow: Code Editor -> Proxy Server -> Toxicity Filter -> LLM
  - Outbound Flow: LLM -> Proxy Server -> Toxicity Filter -> Code Editor
  - Copilot in the code editor does not retain any prompts like code or other context used for the purposes of providing suggestions to train foundational models.
  - Data in Copilot chat is typically retained for 28 days by GitHub Copilot

Slash Commands:

- /fix: Tells you how to fix the code
- /doc: Adds comments for the selected code
- /explain: Gets explanations about the code
- /generate: Generates code to answer the specified question
- /help: Gets help on how to use Copilot chat
- /optimize: Analyzes and improves the runtime of selected code
- /tests: Creates unit tests for the selected code

Context Specific Suggestions:

- @workplace: Provide suggestions based on the entire workspace
- @terminal: Provide suggestions based on the terminal output
- @file: Focus on the content of a specific file
- @directory: Consider the contents of a specific directory

GitHub Copilot Supported Languages:

- Python
- Javascript
- Java
- Typescript
- Ruby
- Go
- C#
- C++
- Copilot provides these languages with exceptional support, but it can assist with many other languages and frameworks as well.

Copilot Free Tier:

- 2,000 Code autocompletions and 50 chat messages per month
- Public Code Filter: If Copilot's code matches existing code in public repos on GitHub it can be blocked out and not shown, this applies to free tier, pro, business and enterprise

Copilot Business & Enterprise:

- Both have User management, Data is excluded from training by default, Enterprise grade security, IP indemnity, Content exclusions, SAML SSO Authentication, Business does not require GitHub Enterprise Cloud but Enterprise does. They both offer usage metrics.
- Enterprise lets you tailor chat conversations to your private codebase
- There are unlimited integrations with copilot extensions in all tiers
- You can build private extensions for internal tooling in all tiers
- You can attach knowledge bases to chat for organizational context only with enterprise licenses
- Things to consider when selecting a copilot pricing plan are: data privacy and security, policy management, data collection and retention, IP Indemnity and data privacy

GitHub Copilot Contractual Protections:

- *IP Indemnity:* The GitHub Copilot Business & Enterprise plans provide legal protection against intellectual property claims related to the use of Copilot suggestions. The matching public code setting must be blocked. If any suggestion from GitHub Copilot is challenged as infringing on third-party IP rights, GitHub assumes legal responsibility.
- *Data Protection Agreement (DPA):* GitHub offers a DPA that outlines the measures taken to protect your data and ensure compliance with data privacy regulations.

- *GitHub Copilot Trust Center:* The trust center provides detailed information about how GitHub Copilot works, including security, privacy, compliance, and intellectual property safeguards.

Common AI Use Cases for Developer Productivity:

- Accelerate learning new programming languages and frameworks
- Minimize Context Switching
- Enhanced Documentation Writing
- Automating the boring stuff
- Personalized Code Completion

Implementing a Measurement Framework:

- Evaluation: During the initial phase of Copilot Adoption, focus on leading indicators such as dev satisfaction and task completion rates. Use API to collect metrics like average daily active users, total acceptance rate, and lines of code accepted.
- Adoption: As GitHub Copilot becomes more integrated into your team's workflow, continue to monitor productivity metrics and enablement indicators. The API can provide insights into user engagement and identify areas where further training is needed.
- Optimization: Once GitHub Copilot is fully adopted, use the REST API for GitHub Copilot usage metrics to fine-tune its impact on the broader organizational goals, such as reducing time to market or improving code quality across the team.
- Sustained efficiency: Continuously evaluate GitHub Copilot's effectiveness as your org evolves. The API allows for ongoing monitoring and adjustment to ensure long term productivity gains.

Introduction to Copilot Business:

- GitHub Copilot Business automatically blocks common insecure code suggestions by targeting issues such as hard-coded credentials, SQL injections, and path injections. It also supports VPN proxy support so it works with VPNs including self-signed certificates, allowing devs to use it in any working environment.
- Any company can quickly purchase Copilot business licenses online and easily assign users, even if they don't use the GitHub Platform for their source code.
- After you enforce your GitHub Copilot Business policy you navigate to your organizations in your profile dropdown menu to enable it

GitHub Copilot Business vs Copilot Pro & Free:

- Business allows you to exclude specific files from GitHub Copilot, have organization-wide policy management, audit logs and increase GitHub Models rate limits

Introduction to Copilot Enterprise:

- Copilot Enterprise comes with Knowledge bases and custom models, chat custom to your codebase, PR summaries, Doc search and summaries using docsets, Code review.
- With Copilot Enterprise you can fine-tune private, custom models, which is built on a company's specific knowledge base and private code.

GitHub Advanced Security

What is GHAS (GitHub Advanced Security)?:

- Application security solution that is embedded right into your workflow to help prevent vulnerabilities and credential leaks without slowing development.

What is Secret Scanning?:

- A crucial security feature within GHAS designed to identify and mitigate the inadvertent exposure of sensitive information, such as API keys and tokens within the source code.
- Operates by searching for predefined patterns and signatures indicative of sensitive information, ensuring that potential security risks are promptly addressed.
- By default, secret scanning looks for highly accurate patterns that have been provided by a GitHub Partner, however, custom patterns can be created for other use cases.
- Secret Scanning includes:
    - *Push Protection:* Proactively prevents secret leaks by scanning code on commit and blocking a push if a secret is present.
    - The ability to easily view alerts and remediate them without ever having to leave GitHub.
- By integrating secret scanning into the dev process, teams can identify and remediate exposed secrets early, reducing the risk of data breaches and ensuring the confidentiality of sensitive data throughout the entire development life cycle.

What is Code Scanning?:

- Code scanning is an integral part of GHAS that analyzes source code for security vulnerabilities and coding errors. It employs static analysis techniques to identify potential issues such as SQL injection, cross-site scripting, and buffer overflows.
- It enhances the overall security of a project by identifying and addressing security vulnerabilities early, before they reach production.
- Code scanning helps minimize the potential impact of security threats, improves code quality, and accelerates the development cycle by reducing the time spent on post-deployment issue resolution.

What is Dependabot?:

- Dependabot is an automated dependency management tool, responsible for keeping project dependencies up-to-date.
- It regularly checks for updates to libraries and frameworks used in a project and automatically opens PRs to update dependencies to their latest, secure versions.
- Dependabot streamlines the process of updating dependencies, ensuring that projects benefit from the latest security patches and improvements.
- With GHAS, Dependabot's functionality is extended to include Dependency Review, allowing you to check for vulnerable dependencies within a PR. This check enables you to address vulnerabilities before they get merged into a shared branch.

How to enable GHAS Features:

- Go to the security tab at the repo level. Then you can enable the different security features.

How to Utilize GHAS to get the most Impact:

- The dependency graph is central to supply chain security. It identifies all upstream dependencies and public downstream dependents of a repository or package.
- You can see your repo's dependencies and some of their properties, like vulnerability information, on the dependency graph for the repository.
- To generate the dependency graph, GitHub looks at a repo's explicit dependencies declared in the manifest and lockfiles.
- Key points about the dependency graph:
    - Includes information on your direct dependencies and transitive dependencies
    - It's automatically updated when you push a commit to GitHub that changes or adds a support manifest or lock file to the default branch.
    - You can see it by going to a repo's main page and then navigating to Insights tab.
    - If you have read access to repo, you can export the dependency graph as a Software Bill of Materials (SPDX-compatible).
    - When a potential vulnerability is detected then you get a Dependabot alert
    - GitHub generally recommends lock files in the repository, because they define the exact versions of the direct and indirect dependencies people are using.

GHAS Alerts:

- GHAS provides holistic visibility into an organization's security posture and the ability to enforce security adoption, enabling precise and effective prioritization and management of security risks.
- *Code Scanning Alerts: CodeQL Analysis Alerts* are generated by CodeQL, GitHub's semantic code analysis engine, these alerts identify potential security vulnerabilities in the codebase. They cover a wide range of issues, including but not limited to SQL injection, cross site scripting and other vulnerabilities.
- *Secret Scanning Alerts: Exposed Secrets Alerts*, these alerts are triggered when potentially sensitive information such as API keys or credentials are identified within the repo's source code. Helps prevent accidental exposure of confidential data.
- *Dependency Alerts*: *Dependabot Alerts*, automatically detects outdated dependencies in a project and creates PRs to update them to the latest, secure versions. Dependabot alerts notify developers about available updates for project dependencies.
- *Security Overview Alerts*: The security overview provides a comprehensive dashboard summarizing the security status of the repository.
- *Third Party Alerts*: You can integrate third-party code analysis tools with GitHub code scanning by uploading data as SARIF files.
- Ignoring a security alert poses significant risks to the project. Vulnerabilities may be exploited by malicious actors, leading to data breaches, service disruptions or other security incidents. Ignoring alerts can also result in increased remediation efforts, potentially impacting project timelines and overall trustworthiness of the software.

- Long-term consequences of ignoring alerts may include reputational damage, regulatory noncompliance, and financial losses. It's crucial for development teams to prioritize and address security alerts promptly to mitigate these risks.
- GHAS provides granular access controls, allowing organizations to define who can view alerts for different security features. This ensures that only authorized personnel, such as security teams and relevant stakeholders, have access to sensitive security information.
- Code scanning & Dependabot alerts can be seen and modified by anyone with the *Write* repository role.
- Secret Scanning alerts can be seen and modified by anyone with the *Admin* repository role.
- Any person or team can be granted access to see and modify all alerts on a repository, regardless of their repository role, by modifying the repo's "Access to alerts" settings

GitHub Advisory Database:

- It is a security vulnerability database inclusive of CVEs and GitHub originated security advisories from the world of open source software.
- Provides a free and open source repository of security advisories
- Enable the community to crowd-source their knowledge about these advisories
- Surface the vulnerabilities in an industry accepted formatting standard for machine interoperability.

Software Bill of Materials (SBOM):

- A formal machine readable inventory of a project's dependencies and associated information (such as version, package identifiers, and licenses).
- You can export the current state of the dependency graph for your repo as a SBOM using the GitHub UI or the REST API
- If a company provides software to the federal government, they will need to provide an SBOM for their product.

Dependabot:

- Automates managing your repository's dependencies. Dependabot keeps your dependencies up to date by informing you of any security vulnerabilities in your dependencies and automatically opens pull requests to upgrade your dependencies to the next available secure version
- For Dependabot to work, the dependency graph must be enabled in a repository.
- Dependabot alerts are generated for vulnerable dependencies when a new advisory is added to the GitHub Advisory Database or the dependency graph for a repository changes.
- Dependabot alerts aren't enabled for public or private repositories by default. To enable them you need to enable both the dependency graph and dependabot.
- Alerts can be set at the level of all repositories you own, all repositories in an organization or all repositories that organizations in your enterprise own.

- Dependabot security updates help you fix the vulnerabilities that Dependabot alerts identify. Version updates help manage different versions of dependent packages.
- Once you've set up Dependabot alerts to notify you of vulnerabilities in your repository, you can enable two related features so that Dependabot automatically opens PRs to try to help with your dependency management:
    - **Dependabot Security Updates** are automated PRs that help you update dependencies with known vulnerabilities.
    - **Dependabot Version Updates** are automated PRs that keep your dependencies updated, even when they don't have any vulnerabilities.
- To reduce the number of pull requests from security updates, there's a grouped security updates capability you can enable to group sets of dependencies together.
- Dependabot is free to use for all users and repositories on GitHub.com. This includes both public and private repositories and enterprise.
- To enable version updates, you need to create a *dependabot.yml* file. This essentially tells Dependabot where to find the manifest or other package definition file.
- The Dependabot.yml file should include the following information:
    - *version* - should be set to 2
    - *registries* - optional if you have dependencies in a private registry
    - *updates* - include an entry for each dependency you want Dependabot to monitor
    - For each package manager include:
        - *package-ecosystem:* specifies the package manager
        - *directory:* specifies the location of the manifest or other definition files
        - *schedule.interval:* specifies how often to check for new versions
- Dependabot updates are not automatically enabled on forks. This prevents fork owners from unintentional version updates for their dependencies.
- To receive a dependabot alert, admins must be watching the repository, have enabled notifications for security alerts or all activity on the repo and not be ignoring the repo.
- GitHub never publicly shares vulnerability info for any repo, this information is only available to repo owners, people with admin permissions, and users who have been granted the appropriate access.
- Dependabot rules: Auto-triage rules that allow you to instruct dependabot to automatically triage dependabot alerts. You can automatically dismiss or snooze certain alerts, or specify the alerts for which you want dependabot to open pull requests.
    - Two types of auto-triage rules for Dependabot: GitHub presets & Custom auto-triage rules
- You can use the GraphQL API with a created OAuth token to retrieve and export Dependabot alert information.
- The preferred method of interaction for GraphQL is the GraphQL explorer via app. The GraphQL API has a single endpoint that doesn't change: https://api.github.com/graphql

Secret Scanning:

- Secret scanning is a GitHub advanced security feature that scans repositories for known types of secrets. Prevents the fraudulent use of secrets that were accidentally committed.

- Secret scanning automatically scans your entire Git history on all branches present in your GitHub repository for any secrets. Additionally, secret scanning scans titles, descriptions and comments in open and closed historical issues, pull requests & GitHub Discussions.
- Push protection prevents secret leaks by scanning for highly identifiable secrets before they're pushed. When a secret is detected in code, contributors are prompted directly in their IDE or CLI with remediation guidance to ensure that the secret isn't inadvertently exposed.
- Push protection is on by default for users for public projects and can't be turned off or configured.
- GitHub offers validity checks for select tokens, this helps remediate exposure to see if a token is still active and when possible whether it was ever active.
- For private repositories with GHAS, secret scanning has a few configurable parameters, such as excluding files from being scanned and configurable notification recipients.
- For private repositories with GHAS, secret scanning has a few configurable parameters such as excluding files from being scanned and configurable notification recipients.
- You can enable secret scanning at a repository level or organization level.
- You may not want to include test files or files that contain generated content to avoid generating false alerts. You can create a .github/secret_scanning.yml file in your repo that excludes some directories from being scanning.You can use special characters such as * to filter paths. If there are more than 1,000 entries in paths-ignore, secret scanning will only exclude the first 1,000 directories from scans. If the file is larger than 1MB the whole file will be ignored.
    - Ex: paths-ignore:
      - "foo/bar/*.js"
- Alerts can be filtered with bypassed, validity, secret type, & provider
- Once a secret has been committed into a repository, you should consider the secret compromised. GitHub recommends the following actions for compromised secrets:
    - For a compromised GitHub PAT, delete the compromised token, create a new token and update any services that use the old token.
    - For all other secrets, first verify that the secret committed to GitHub is valid. If so, create a new secret, update any services that use the old secret, and delete the old secret.
- Secret scanning supports up to 500 custom patterns for each organization or enterprise account, and up to 100 custom patterns per private repository.

Code Scanning:

- Code scanning uses CodeQL to analyze the code in a GitHub repository to find security vulnerabilities and coding errors. Code scanning is available for all public repositories, and for private repositories owned by organizations where GHAS is enabled.
- CodeQL is the code analysis engine GitHub developed to automate security checks. You can analyze your code using CodeQL and display the results as code scanning alerts.
- CodeQL treats code like data, allowing you to find potential vulnerabilities in your code with greater confidence than traditional static analyzers.

- You generate a CodeQL database to represent your codebase, then run CodeQL queries on that database to identify problems in the codebase.
- CodeQL supports the following languages:
    - C/C++
    - C#
    - Go
    - Java/Kotlin
    - Java/Typescript
    - Python
    - Ruby
    - Swift
- Instead of running code scanning in GitHub you can perform analysis elsewhere and then upload the results. You can upload Static Analysis Results Interchange Format (SARIF) files generated outside GitHub or with Actions to see code scanning alerts from third party tools in your repository.
- To use GitHub Actions to upload a third-party SARIF file to a repository you'll need a GitHub Actions workflow.

CodeQL:

- CodeQL analysis relies on extracting relational data from your code and using it to build a CodeQL database. These databases contain all of the important information about a codebase.
- You can use the CodeQL CLI standalone product to analyze code and to generate a database representation of a codebase. After the database is ready, you can query the database or run a suite of queries to generate a set of results in Static Analysis Results Interchange Format (SARIF).
- A CodeQL database is a single director that contains all of the data that's required for analysis. This data includes relational data, copied source files and a language specific database schema that specifies the mutual relations in the data.
- Once the code is extracted to a database you can analyze it by using CodeQL queries.
- You can use CodeQL queries in code-scanning analysis to find problems in your source code and to find potential security vulnerabilities.
- The basic CodeQL query structure has the file extension **.ql** and contains a **select** clause.
- The syntax of the declarative, object-oriented query language (QL) is similar to SQL, but the semantics are based on datalog.

GHAS Administration:

Feature	Public repository	Private repository without Advanced Security	Private repository with Advanced Security
Code scanning	Yes	No	Yes
Secret scanning	Yes (limited functionality only)	No	Yes
Dependency review	Yes	No	Yes
Security Overview	No	No	Yes

- If you enable GHAS in your organization, committers to the organization repositories will use seats on your GHAS license.

Setting Security Policies:

- A security policy defines what it means for a system organization or other entity to be secure, and limits permissions to support that definition.
- **SECURITY.md** is a way to give people instructions for reporting security vulnerabilities in your project, you can add a SECURITY.md file to your repository root. It should include a list of supported versions of the project and a way to report a security vulnerability. It might also include information about the project's compliance with key privacy and security laws, technologies that administrators and stakeholders use to secure information, and known risks.
- Fundamental principles and best practices for adjusting permissions and policies to remain secure, compliant, and adaptable to changing needs:
    - Least privilege: Grant users the minimum permissions necessary.
    - Risk-Based Adjustments: Regularly assess and adjust policies based on evolving security risks
    - Periodic Reviews: Schedule reviews of permissions and policies to ensure they remain effective.

    - Customizability: Allow for flexibility to accommodate the specific needs of different teams or projects.
    - Documentation: Maintain clear docs of policy changes to support audits and continuous improvement.

Repository Rulesets:

- Repo Rulesets provide a powerful way to control how users can interact with certain branches and tags in a repo. By creating rulesets, you can define a named list of rules that govern various actions, such as pushing commits, deleting or renaming tags, and more.
- Rulesets work alongside branch-protection and tag-protection rules, allowing you to have fine-grained control over your repository's behavior.
- You can control things like who can push commits to a certain branch or who can delete or rename a tag.
- For example, you could set up a ruleset for your repo's *feature* branch that requires signed commits and blocks force pushes for all users except repo admins. There's a limit of 75 rulesets per repo.
- Rulesets have the following advantages over branch and tag protection rules:
    - Unlike protection rules, multiple rulesets can apply at the same time, so you can be confident that every rule targeting a branch or tag in your repository will be evaluated when someone interacts with that branch or tag.
    - Rulesets have statuses, so you can easily manage which rulesets are active in a repository without needing to delete rulesets.
    - Anyone with read access to a repository can view the active rulesets for the repository. This means a developer can understand why they have hit a rule, or an auditor can check the security constraints for the repo, without requiring admin access to the repo.
- It's set in Settings > Code and automation > Rules > Rulesets
- Rulesets are combined together and the most restrictive rules will be applied if there is overlap

Exam Prep:

- CodeQL is a code analysis tool.
- In the context of security, *shifting left* means adopting security practices early in the development cycle.
- You can use repository security advisories to privately discuss, fix, and publish information about security vulnerabilities in your public repository.
- Dependabot helps you keep the repository dependencies up to date.
- The GitHub Advisory Database is a curated list of security vulnerabilities found in open source projects
- The following features are available for FREE for both public and private personal repositories: Security Policy, Security advisories, Dependabot alerts and security updates, & Dependabot version updates.
- Secret scanning scans your repository for secrets such as private keys or tokens.

- Secret scanning scans the entire git history on all branches in the repository. It also scans the titles, descriptions, and comments in open and closed historical issues.
- The Secret scanning partner program is a program where service providers can partner with GitHub so that the format of their secrets can be recognized by GitHub secret scanning.
- Public repositories owned by personal users as well as public repositories owned by organizations can use secret scanning for free.
- To prevent commits containing cloud provider credentials from being pushed to GitHub you must enable secret scanning push protection for your repository or organization.
- When GitHub identifies a secret from a partnered service provider, it notifies the service provider about the leaked secret. The partner can take actions upon receiving notification from GitHub about a leaked secret, such as revoking the secret and informing the owner of the compromised secret. It is a program where service providers can provide GitHub with the regex patterns of secrets that they issue so GitHub secret scanning can recognize them.
- You can exclude certain directories or files from secret scanning by creating a secret_scanning.yml file and including paths that should not be scanned.
- You can dismiss GitHub secret scanning alerts that are fake and used in tests by closing the secret scanning alerts with the *Used in tests* close reason.
- If you have accidentally committed your GitHub PAT to a public repo you should consider the token compromised and delete it immediately.
- When a new secret pattern is added or updated in the GitHub secret scanning partner program GitHub will run a scan of all historical code content in public repositories with secret scanning enabled.
- When a new secret is pushed and detected in a repository, repository administrators, security managers, users with custom roles with read/write access, organization owners and enterprise owners, if they are administrators of repositories where secrets were leaked and commit authors.
- When GitHub runs a scan of all historical code in enterprise repositories GitHub notifies the enterprise owners and security managers, even if no secrets are found. GitHub notifies repo admins, security managers, and users with custom roles with read/write access whenever a secret is detected in a repo.
- Github does not use the same set of secret scanning patterns for both user alerts and push protection alerts. There are three different sets of secret scanning patterns. The three different sets of secret scanning patterns GitHub maintains are Push protection patterns, user alert patterns, and partner patterns.
- You can enable push protection for yourself in your personal GitHub account settings if you are contributing to public repos.
- Dependabot alerts tell you that your repository uses a package that is insecure.
- The Dependency graph knows what dependencies your project is using because it derives dependencies automatically from manifests and lock files committed to the repo. Dependencies can also be manually added using the dependency submission API.
- You can export the github dependency graph in SPDX format.

- Dependabot Alerts partially rely on the GitHub Advisory Database, when GitHub identifies a vulnerable dependency, they generate a Dependabot alert and display it on the security tab for the repo, to enable dependabot alerts you first need to have dependency graph enabled on your repo.
- The default CodeQL analysis set up in GitHub automatically chooses languages to analyze, query suite to run, and events that trigger scans.
- The top-level keys that are required in the *dependabot.yml* file are *version* and *updates.*
- Use *paths-ignore:* key to exclude directories from secret scanning alerts in GitHub
- The max number of custom patterns that can be defined for secret scanning is 100 for repositories and 500 for orgs/enterprises.
- Code scanning allows you to find, triage and prioritize fixes for new and existing problems in your code.
- You can configure your GitHub repository to run CodeQL analysis on a schedule by using the default CodeQL analysis setup. Also, by creating a GitHub Actions workflow with a *schedule* trigger.
- CodeQL uses a different extractor for each programming language and it creates separate databases for each programming language.

GitHub Actions

What are GitHub Actions?:

- GitHub Actions are a flexible way to automate nearly every aspect of your team's software workflow. You can automate testing, continuously deployments, review code, manage issues and pull requests, and much more. The best part, these workflows are stored as code in your repository and easily shared and reused across teams.
- Can be triggered each time developers check new source code into a specific branch, at timed intervals, or manually.
- There are three types of GitHub actions: container actions, JavaScript actions, and composite actions.
    - **Container actions:** The environment is part of the action's code. These actions can only be run in a Linux environment that GitHub hosts. Support many different languages.
    - **JavaScript actions:** The environment is not included in the code.You have to specify the environment to execute these actions. You can run these in a VM (virtual machine) in the cloud or on-prem. Support Linux, MacOS, and Windows environments.
    - **Composite actions:** Allow you to combine multiple workflow steps within one action. You can use this feature to bundle together multiple run commands into an action, then have a workflow that executes the bundled commands as a single step using that action.
- This is a good demo we can use to show customers how to add comments to PRs: https://github.com/skills/hello-github-actions. Can modify this as needed to find a more real world use case.

What is a GitHub Actions workflow?:

- It is a process that you set up in your repository to automate SDLC tasks. With a workflow, you can build, test, package, release and deploy any project on GitHub.
- To create a workflow, you add actions to a *.yml* file in the *.github/workflows* directory in your repository.
- The different attributes:
    - on: This is a trigger to specify when this workflow will run. It can trigger on a push event, pull_request, manually or more.
    - jobs: A workflow must have at least one job. A job is a section of the workflow associated with a runner.
    - runs-on: A runner can be GitHub-hosted or self-hosted, and the job can run on a machine or in a container.

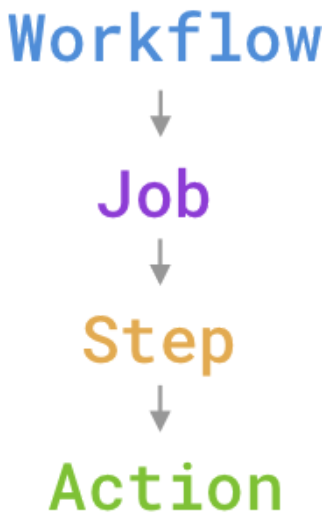
GitHub Hosted vs Self-Hosted Runners:

- If you use a GitHub-hosted runner, each job runs in a fresh instance of a virtual environment. They offer a quicker and simpler way to run your workflows, albeit with limited options.

- With a self-hosted runner you need to apply the self-hosted label, its operating system, and the system architecture. They are a highly configurable way to run workflows in your own custom local environment.
- You can run self-hosted runners on premises or in the cloud. You can also use them to create a custom hardware configuration with more processing power or memory. This will help to run larger jobs, install software available on your local network, and choose an operating system not offered by GitHub hosted runners.

<u>The Components of GitHub Actions:</u>

- There are several components that work together to run tasks or jobs within a GitHub Actions workflow.
- Event -> Workflow -> Job -> Step -> Action. An event triggers the workflow, which contains a job. This job then uses steps to dictate which actions will run within the workflow.
- **Workflows:** An automated process that you add to your repository. A workflow needs to have at least one job, and different events can trigger it. You can use it to build, test, package, release, or deploy your repository's project on GitHub.
- **Jobs:** The job is the first major component within the workflow. A job is a section of the workflow that will be associated with a runner. A runner can be GitHub hosted or self hosted, and the job can run on a machine or in a container. The runner is specified at the job level with the *runs-on:* attribute.
- **Steps:** A step is an individual task that can run commands in a job. For example, a step can use actions like *actions/checkout@v4* to checkout the repository.
- **Actions:** The actions inside your workflow are the standalone commands that are executed. These standalone commands can reference Github actions such as using your own custom actions, or community actions like the one we use in the preceding example. You can also run commands here such as *run: npm install -g bats* to execute a command on the runner. Use specific versions of actions to avoid breaking workflows.



Configure Workflows to Run Schedule Events:

- The schedule event allows you to trigger a workflow to run at specific UTC times using CRON syntax. The cron syntax has five * fields and each field represents a unit of time.
- * minute (0-59)
  * hour (0-23)
    * day of the month (1-31)
      * month (1-12 or JAN - DEC)
        * day of the week (0 - 6 or SUN - SAT)
- For example if you wanted to run a workflow every 15 minutes:

```
on:
  schedule:
    - cron: '*/15 * * * *'
```

- For example if you wanted to run a workflow every Sunday at 3:00 AM:

```
on:
  schedule:
    - cron: '0 3 * * SUN'
```

- The shortest interval you can run scheduled workflows is once every five minutes. They run on the latest commit on the default or base branch.

Configure Workflows to run for Manual Events:

- You can manually trigger a workflow by using the *workflow_dispatch* event. This event allows you to run the workflow by using the GitHub REST API or by selecting the Run workflow button in the Actions tab within the repo.
- Using *workflow_dispatch*, you can choose on which branch you want the workflow to run, as well as set optional *inputs* within it so that GitHub will present as form elements in the UI.
- In addition to *workflow_dispatch* you can use the GitHub API to trigger a webhook called *repository_dispatch*. This event allows you to trigger a workflow for activity that occurs outside of GitHub. It essentially serves as an HTTP request to your repository asking GitHub to trigger a workflow off an action or webhook.

Configure Workflows to run for Webhook Events:

- You can configure a workflow to run when specific webhook events occur on GitHub.
- Within your workflow file, you can access context information and evaluate expressions.
- You just need to use the *if* conditional

GitHub Script

- GitHub Script is an action that provides an authenticated octokit client and allows JavaScript to be written directly in a workflow file. It runs in node.js, it is a workflow action that provides you with access to the GitHub API from within your GitHub Actions.
- Octokit is the official collection of clients for the GitHub API.
- GitHub Script actions fit into a workflow like any other action. You can even mix them in with existing workflows

GitHub Packages:

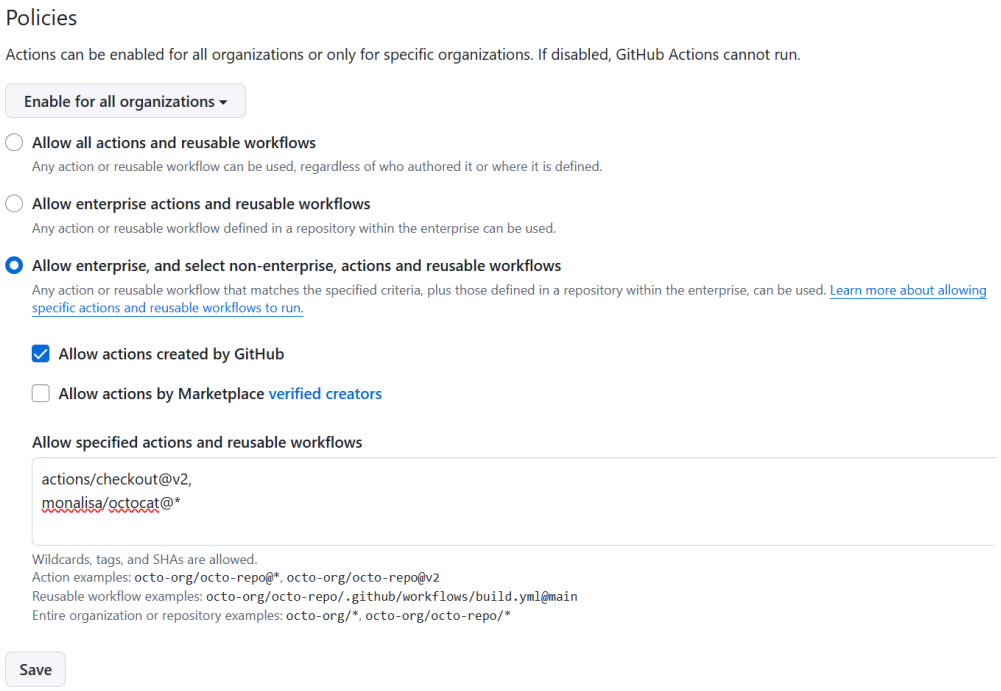
- GitHub Packages is a package-management service that makes it easy to publish public or private packages next to your source code.
- GitHub Packages use your GitHub identity to authenticate you.
- Docker is an engine that allows you to run containers.
- Containers are packages of software that can reliably run in different environments. Containers include everything needed to run the application.
- A Dockerfile is a text document that contains all the commands and instructions necessary to build a Docker Image.
- A Docker Image is an executable package composed of code, dependencies, libraries, a runtime, environment variables, and configuration files. This can get pushed to Packages.
- A Docker container is a run time instance of a Docker Image.
- The way you authenticate into your package manager will depend on your project's ecosystem. Whichever ecosystem you're working with you'll need your GitHub username, A PAT, and the GitHub Packages endpoint for your package ecosystem.

Create and Publish custom GitHub Actions:

- GitHub Actions allow you to create individual, custom actions by writing code that interacts with your repository in any way you'd like
- There are three different types of actions: Docker container actions, JavaScript actions, and composite run steps actions.
- *Docker Container Actions:* They package the environment with the GitHub Actions code. This means that the action runs in a consistent and reliable environment because all of its dependencies are within that container.
- *JavaScript Actions:* Run directly on the runner machine, and separate the action code from the environment that's used to run the action. It is simplified and can execute faster than actions within a Docker container.
- *Composite Run Steps Actions:* Allow you to reuse actions by using shell scripts. You can even mix multiple shell languages within the same action. If you have many shell scripts, you can now easily turn them into an action and reuse them for different workflows.
- *Inputs:* Inputs are the parameters that allow you to specify data that the action expects to use during its runtime. GitHub stores these parameters as environment variables. These are optional.
- *Outputs:* The parameters that allow you to declare data. Actions that run later in a workflow can use the output data that was declared in a previously run action.
- *Runs:* Your action needs to have a *runs* statement that defines the command necessary to execute your action.
    - Runs for Docker Actions: configures the image the Docker action uses.
    - Runs for JavaScript Actions: have to tell it what application to use to execute the code and where the action code is located.
    - Runs for Composite Step Actions: have to explicitly say "composite" and then give it the steps you want to run.

Managing GitHub Actions in the Enterprise:

- GitHub Enterprise Cloud is a cloud-based solution tailored for organizations that need the flexibility and scalability of GitHub's infrastructure while maintaining enterprise-grade security and compliance. It offers features such as centralized user management, data residency options, and integration with identity providers like SAML and SCIM for single sign on and user provisioning.
- Standard User Model: Users manage their own account
- Enterprise Managed User (EMU): This is the model designed for organizations requiring centralized user account management.
- GitHub Enterprise Server is a self-hosted version of GitHub designed for organizations that require complete control over their GitHub instance.
- At the Enterprise Level:
    - Configure a GitHub Actions use policy: Navigate to your enterprise account and then to Policies > Actions in the side bar. Here you can enable only specific actions to be used within your enterprise. This is potentially a good safe guard if you don't want to allow potentially malicious actions or want to have a verification process for actions.



- At the organization level:
    - Document Corporate Standards: As best practice, we recommend you document some kind of standards for how to best maintain things.
    - Reusable templates help standardize and streamline development across multiple repositories, reducing redundancy and improving maintainability. Reusable workflows are defined in a separate repository and then can be referenced in multiple projects.
        - They are like standardized, centralized CI/CD pipelines across repositories.

        - A reusable workflow is stored in .github/workflows/ and uses the workflow_call trigger as opposed to workflow_dispatch
        - Once defined the reusable workflow can be used in any repository via the *uses:* keyword.
        - Benefits: Ensures all repositories follow the same CI/CD structure, reduces redundancy and maintenance overhead, allows for centralized updates without modifying each repository.
- Enterprise administrators need to strike a balance between giving developers flexibility and ensuring security and governance over automation workflows.
- If an organization hosts reusable GitHub Actions in a dedicated repository, access should be limited to authorized users.
    - Use branch protection rules to prevent unauthorized modifications
    - Require pull request approvals for updates to actions
    - Restrict write access using Role based access control
- Organizations should control which external actions can be used.
- Risks of using all external actions: Malicious Code Execution: An untrusted external action could introduce vulnerabilities, Dependency Tampering: A third party action may introduce supply chain attacks, Secrets Exposure: External actions may inadvertently log secrets.
- Best practice to configure external action restrictions
- Risks of self-hosted runners:

Risk	Description	Mitigation
Unauthorized Access	Attackers could hijack runners to execute malicious actions.	Restrict access using **IP allow lists**.
Secret Exposure	Sensitive credentials may be leaked to compromised runners.	Store secrets in **GitHub Secrets Management** instead of environment variables.
Runner Compromise	If runners are not isolated, workflows from different teams could interfere.	Use **ephemeral runners** that reset after each job.

- Enable Audit logs for GitHub Actions, Org Settings -> Security -> Audit Log, filter logs by actions can see who ran workflows, which actions were executed and what permissions workflows used.
- Good best practices on managing reusable actions: https://learn.microsoft.com/en-us/training/modules/manage-github-actions-enterprise/manage-leverage-reusable-actions

- Best Practices for Distributing Actions in an Enterprise:

Best Practice	Recommendation
Standardization	Maintain a **single repository** for reusable actions with **clear documentation**.
Versioning	Use **semantic versioning** (`v1.0.0`, `v2.0.0`) and avoid `latest` for stability.
Security Reviews	Require **code reviews and automated security scans** before publishing actions.
Access Control	Restrict **who can modify and use** internal actions via repository settings.
Regular Updates	Maintain **a changelog** and notify users of updates to prevent breaking changes.

- GitHub-Hosted vs Self-Hosted Runners:

Feature	GitHub-hosted runner	Self-hosted runner
Setup & maintenance	No setup required; GitHub manages everything	User must install, configure, and maintain
Scalability	Auto-scales dynamically	Must manually provision additional runners
Security	High security; fresh virtual environment for each job	Requires manual security hardening
Customization	Limited; pre-installed tools only	Fully customizable; user can install any dependencies
Performance	Standardized compute resources	Can use high-performance hardware
State persistence	Resets after every job	Can persist data between jobs
Cost	Free for public repos; limited free usage for private repos	No GitHub costs, but requires infrastructure investment
Network access	No direct access to internal networks	Can access internal/private networks
Use case	Best for general CI/CD, automation, and open-source projects	Best for enterprise environments, secure builds, and large workloads

Exam Prep:

- The purpose of the ‘env’ keyword in a GitHub Actions workflow file is to define environment variables for the entire workflow.

- To set up a workflow that runs a container action upon code pushes to the repository, you should define the “on:push” trigger and include necessary steps that utilize the container action.
- Steps are responsible for executing individual commands within a job.
- The primary purpose of the ‘uses’ statement in a GitHub Actions workflow file is to include a specific action or command in a step.
- Container actions require you to specify the execution environment and support multiple operating systems like Linux, macOS, and Windows.
- When reviewing an open-source GitHub Action for use in your project, you should review the action’s action.yml file for inputs, outputs, and code functionality.
- You can use the *Uses* attribute in the path to the Docker image for a container action
- If your organization needs to automate the deployment of a web application using GitHub Action you can configure a container action to be executed automatically when new code is pushed to the repository by defining the container action in an action.yml file and set the trigger to ‘on:push’ in the workflow file.
- If your team wants to automate the process of building Docker images using GitHub actions whenever there is a push to the repository you can use container actions, as they encapsulate the environment within Docker images.
- When a workflow creates something other than a log entry, the product is called an artifact.
- Instead of downloading dependencies over and over again, you can cache them to make your workflow run faster and more efficiently. Caching dependencies will help speed up the time it takes to recreate dependency files.
- You can reclaim used GitHub Actions storage by deleting artifacts before they expire on GitHub. Once an artifact is deleted, it can’t be restored.
- The default artifact retention period is 90 days but this can be changed at the repo, org and enterprise level. You can also set it within the workflow for individual use cases.
- GITHUB_TOKEN permissions passed from the called workflow can only be downgraded by the called workflow.
- The different permission levels you can assign to GITHUB_TOKEN in the permissions block are none, read, write.
- You can use permissions to modify the GITHUB_TOKEN permissions on the Job and Workflow level.
- GitHub Actions are free for public repositories.
- Cloning a repo is not a valid event that would trigger a workflow.
- Workflows can be run one or multiple jobs at a time, workflows have to be defined in the .github/workflows directory, workflows can be triggered manually, by an event or run on a schedule
- A workflow must contain the following components: One or more jobs and one or more events that will trigger the workflow.
- repository_dispatch is an event that is triggered by a webhook action from outside of the repository.
- Workflows are defined in the yaml format

- You should store sensitive data such as passwords or certificates that will be used in workflows in secrets.
- In a workflow with multiple jobs the default behavior is all jobs run in parallel.
- If job B requires job A to be finished you have to use the needs keyword in job B to create this dependency.
- In a workflow with multiple jobs, if job A fails then the jobs that are dependent on job A are skipped.
- You can use the matrix strategy to parallelize entire workflows.
- You can define a matrix job like this:

```
jobs:
 example_matrix:
      strategy:
        matrix:
          version: [10, 12, 14]
          os: [ubuntu-latest, windows-latest]
```

- You can access the matrix variables in a matrix strategy job using the ***matrix*** context.
- When using the *pull_request* and *pull_request_target* events, you can configure the workflow to run only when targeting the *prod* branch by using the *branches* filter.
- When using *push* event trigger filters you can use glob patterns to target multiple branches.
- workflow_dispatch allows you to manually trigger a workflow from the GitHub UI.
- The possible types of an input variable for a manually triggered workflow are number, string, choice, boolean, and environment.
- A workflow that has only workflow_dispatch event trigger can be triggered using GitHub's REST API.
- To stop a workflow from running temporarily without modifying the source code you should use the *disable workflow* option in GitHub Actions.
- The *activity types* of an event are used for limiting workflow runs to specific activity types using the *types* filter.
- If you want to create a reusable workflow *CI* that runs some quality checks, linting and tests on code changes. You should use a workflow_call trigger to allow reusing it in other workflows.
- The valid use cases for using defaults are using defaults.run on workflow level to set default shell (e.g. bash) for an entire workflow and using defaults.run on job level to set default working-directory for all steps in a single job.
- You can use concurrency to ensure that only a single job or workflow using the same concurrency group will run at a time.
- You can use the always() conditional to always run a job regardless of whether the jobs it needs are successful.
- github.repository is in "org/repo-name" format
- Not all steps run actions, but all actions run as a step.
- For any action published in GitHub Marketplace you can often use it in multiple versions, the most stable and secure approach is to reference the commit SHA.

- To prevent a job from failure when one of the steps fails you can include a continue-on-error flag in the failing step.
- If you define a matrix job, you can limit the matrix to run a maximum of 2 jobs at a time by setting jobs.example_matrix.strategy.max-parallel to 2.
- To echo an environment variable all you have to do is echo “$env_var”
- A reusable workflow can call another reusable workflow.
- All branches can restore caches created on the default branch.
- You cannot access artifacts that are created in a different workflow run.
- You should use artifacts to store coverage reports or screenshots generated during a workflow that runs automated testing for a repository.
- You can upload more than one file at a time when using actions/upload-artifact
- In job *deploy* if you want to access binaries (containing your application) that were created in job *build* you should upload the binaries in *build* and download them in *deploy*.
- You can create a dependency on another job by using the *needs* key word.
- Secrets and configuration variables can be scoped to an environment in a repository, the entire organization, or selected repositories in an organization, and a single repository.
- Secrets cannot be scoped to a specific job in a workflow or a specific workflow in a repo, or multiple repositories that do not share an organization/enterprise.
- Because of latency to build and retrieve the container, Docker container actions are slower than Javascript actions.
- When creating a custom GitHub action the code can live in a separate repo, the action metadata (name, description, outputs, or required inputs) must be defined in the action.yml or action.yaml file in the action repository.
- When you re-run a workflow it runs with the same code, regardless if you pushed new code.
- You can require manual approvals by a maintainer by using deployment protection rules.
- Each job in a workflow can reference a single environment.
- Storing long lasting access keys in GitHub Secrets like accessing resources in cloud providers is not recommended in case of any security leaks or attacks such as script injection instead it is recommended to use OIDC.
- OpenID Connect (OIDC) is an authentication protocol built on top of OAuth 2.0, which provides a way for clients to verify the identity of end users, and for end users to obtain basic profile info about themselves. It is the safest and recommended way to authenticate with cloud providers. (Good best practice)
- Workflows on pull requests to public repositories from forks from some outside contributors will not run automatically, and might need to be approved first.
- GITHUB_ACTOR is an environment variable that contains the name of the person or app that initiated the workflow run.
- The default environment variables in GitHub Actions are GITHUB_WORKFLOW, GITHUB_ACTOR & GITHUB_REPOSITORY.
- If you declare a secret in repository scope, it overwrites the secret that is at the org level.
- The correct way to print a debug message is: echo “::debug::Watch out here!”

- Organizations that are using Enterprise Server can enable automatic syncing of third party GitHub Actions hosted on GitHub.com to their GitHub Enterprise Server Instance by using GitHub Connect.
- You can find network connectivity logs for a GitHub self-hosted runner in the _diag folder directly on the runner machine.
- You can validate that your GitHub self-hosted runner can access all required GitHub services by using a GitHub provided script on the runner machine.
- Secrets cannot be directly referenced in if conditionals
- You can use the GitHub API to download workflow run logs by doing:
  GET /repos/{owner}/{repo}/actions/runs/{run_id}/logs
- You can use the GitHub API to create or update a repository secret by doing:
  PUT /repos/{owner}/{repo}/actions/secrets/{secret_name}
- Self Hosted Runners, Configuration Variables, Workflow Templates and Secrets can be reused within a GitHub Organization.
- The maximum number of reusable workflows that can be called from a single file is 20.
- Each workflow is composed of one or more job which is composed of one or more step, and each step is an action or a script.
- Scheduled workflows run on the latest commit on the repository default branch.
- You can reuse a defined workflow in multiple repositories by using workflow templates and by defining the workflow in a central repository.
- You can ensure a job runs only on a specific branch by using the branches filter.
- The keyword *env* lets you define environment variables in a GitHub Actions workflow.
- The purpose of the *with* keyword is to specify input parameters for an action.
- By using a multiline string with | you can run multiple commands in a single step.
- You can cache dependencies to speed up workflow execution by using the actions/cache action.
- The *matrix* keyword allows you to define multiple job configurations to run in parallel.
- The *concurrency* keyword can be used to limit the number of concurrent jobs running in a GitHub Actions workflow.
- The default timeout for a GitHub Actions job is 360 minutes.
- You can specify the operating system for a job in GitHub Actions by using the runs-on keyword.
- To reference a GitHub Secret in a workflow you use ${{ secrets.SECRET_NAME}}
- The default shell used on Windows runners is powershell.
- You can add a self-hosted runner to an organization, repository, and enterprise.
- RUNNER_OS is the default environment variable that contains the operating system of the runner executing the job.
- For secrets larger than 48 KB it is recommended to encrypt and store secrets in the repository but keep the decryption passphrase as a secret.
- Status check functions in GitHub Actions: success(), always(), cancelled() and failure()
- If you have always() it will always run a job even if all else fails.
- github.event holds context about the event that triggered a workflow run.

- Path filters in GitHub Actions allow you specify when the workflow runs essentially, it helps you control when your actions run based on the location of the files or folders that have been modified.
- In GitHub Actions, if you define both branches and paths, the workflow will only run when both branches and paths are satisfied.
- Treat environment variables as case sensitive.
- restore-keys parameter in actions/cache in GitHub Actions provide alternative keys to use in case of a cache miss.
- In order to enable step debug logging you would set the Actions_Step_Debug. It is obvious in the name.
- timeout-minutes keyword in a step limits the execution time for an individual step.
- At the org level there is a .github repo that's where you can store workflow-templates.
- If you want to be notified when a comment is created on an issue within a repo. The issue_comment event trigger should be used within the workflow configuration.
- You can delete workflow runs when a workflow run has been completed and when a workflow run is two weeks old.
- Only repo admins can bypass configured deployment protection rules to force deployment (by default)
- If any of the following keywords are in the commit message or in the title of a PR the workflow run is skipped: [skip ci], [ci skip], [no ci], [skip actions], [actions skip]
- If *runs.using* has docker as value it is a container action.
- post-entrypoint allows you to specify a cleanup script in a container action.

GitHub Administration

A lot of the information overlaps with the other exams so this section was kept short intentionally and mostly includes info that wasn't covered in the other sections.

GitHub Administration Hierarchy:

- GitHub Administration is basically carried out at 3 levels: team, organization, enterprise
- In GitHub, each user is an organization member that you can add to a team.
- You can create teams in your organization with cascading access permission. A team is a useful substructure for refining repository permissions on a more granular level and enabling communication and notification between team members.

How Does GitHub Authentication Work?:

- Setting up and controlling users' authentication is one of the most common admin tasks performed by organization owners.
- There are several options for authentication with GitHub.
    - The most basic is Username & Password. It's known as the basic HTTP authentication scheme. In recent years basic authentication has proven to be too risky when dealing with highly sensitive information. So it is not recommended.
    - Personal Access Tokens (PAT) are an alternative to using passwords for auth to GitHub when using the GitHub API or command line. Users can generate a token via the GitHub's settings options, and tie the token permissions to a repo or org.
    - SSH Keys: As an alternative to using PATs users can connect and authenticate to remote servers and services via SSH with the help of SSH keys. SSH keys eliminate the need for users to supply their username and PAT for every interaction.
    - Deploy Keys: Are another type of SSH Key in GitHub that grants a user access to a single repo. They are read-only by default, but you can give them write access when adding them to a repo.
- GitHub offers Two-factor authentication, as an extra layer of security used when logging into websites or apps.
- GitHub also offers SAML SSO. If you centrally manage your user's identities and applications with an IdP, you can configure SAML SSO to protect your organization's resources on GitHub.
- LDAP is also popular for GitHub Enterprise Server. LDAP is a popular application protocol for accessing and maintaining directory information services. Can use it to integrate with Active Directory.

Five Repository Level Permissions:

- **Read:** Recommended for non-code contributors who want to view or discuss your project. Good for anyone that needs to view the content but doesn't need to make changes.

- **Triage:** Recommended for contributors who need to proactively manage issues and pull requests without write access. Good for some project managers who manage tracking issues.
- **Write:** Recommended for contributors who actively push to your project. Standard permission for most developers.
- **Maintain:** Recommended for project managers who need to manage the repo without access to sensitive information or destructive actions.
- **Admin:** Recommended for people who need full access to the project, including sensitive and destructive actions like managing security or deleting a repo.
- GitHub applies the highest level of access granted to the user. For example, if a user has Read access through a team but also Write access directly assigned, they will effectively have write permissions.

Managing Enterprise Access, Permissions & Governance:

There are multiple levels of permission at the Organizational level:

- **Owner:** Organization owners can do everything that organization members can do, and they can add or remove other users to and from the organization. This role should be limited to no less than two people in the org.
- **Member:** Org members can create and manage organization repositories and teams.
- **Moderator:** Org moderators can block and unblock nonmember contributors, set interaction limits, and hide the comments in public repositories that the organization owns.
- **Billing Manager:** Organization billing managers can view and edit billing information.
- **Security Managers:** Organization security managers can manage security alerts and settings across your organization. They can also read permissions for all repositories in the organization.
- **Outside Collaborator:** Outside collaborators such as a consultant or temp employee can access one or more organization repositories. They aren't explicit members of the organization.

Pros & Cons of Single vs Multiple Organizations:

- **Pros of Single Organization:**
  - Simplified Management: Centralized control of permissions and policies.
  - Consistency: Uniform application of rules and streamlined collaboration.
  - Resource Sharing: Easier asset sharing across teams.
  - Cost Efficiency: Reduced overhead in administrative tooling and licensing.
- **Cons of Single Organization:**
  - Limited Flexibility: One size fits all policies might not suit all teams.
  - Security Risks: A single breach could impact the entire org.
  - Scalability Issues: Managing permissions can become complex as the org grows.
- **Pros of Multiple Organizations:**
  - Tailored Policies: Customize permissions to fit the specific needs of each team.
  - Enhanced Isolation: Limits the impact of a security breach to a single organization.

    - Decentralized Administration: Teams can manage their own policies and permissions.
- **Cons of Multiple Organizations:**
    - Increased Complexity: More organizations means more administrative overhead.
    - Redundancy: Potential duplication of settings and management efforts.
    - Inter-Org collaboration: May require extra tools or processes for cross-organization projects.

EMUs:

- In GitHub Enterprise Cloud, EMUs ensure that authentication happens strictly through your Identity Provider (IdP). This model restricts access to enterprise managed accounts only. It also enforces centralized control over identity, credentials, and session policies.
- Using EMUs you can connect your GitHub Enterprise Account to an identity provider (IdP); either Microsoft Entra ID or Okta.
- You can centrally manage the user provisioning lifecycle and create a true SSO experience for your members and contractors.

	GitHub Enterprise Managed Users (EMU)	GitHub Enterprise Account
Ownership	Created for and controlled by the identity provider.	Multiple authorization and authentication initiatives required in GitHub to ensure secure enterprise access.
Membership	Automated user provisioning syncs IdP group membership with GitHub teams.	Administrators manage user membership on teams in the enterprise's organizations on GitHub.
Policies	You can create policies at the enterprise level that apply to all users on the account.	Users are required to confirm compliance with GitHub user policies.
Audit log	Enterprise owners can audit all of the managed users' actions on GitHub.com	Only audit log activity that happens in the enterprise account is available for your review.
Privacy	No public repositories on EMU-enabled enterprises.	Enterprise users can create public repositories.
SSO	True single sign-on experience for your users.	Multiple authentication steps required for your users to sign on.
Metadata	The enterprise controls user GitHub metadata. Managed users can't modify it. This metadata includes email addresses, display names, and usernames.	Enterprise users create their own account, provide email, and choose their GitHub handle.

Exam Prep:

- The team maintainer permission is the required permission to add and remove organization members to and from a team.

- You can use SSO to ensure everyone who is signed onto the corporate network can access GitHub without a second sign in.
- The write repository permission will allow contributors to actively push changes to the repo.
- The team maintainer can add or remove team members.
- The triage permission level is best for project managers who need to triage and organize issues without contributing code.
- Centralized Identity Management is a benefit of integrating AD for team synchronization.
- Organization Owners can exclusively manage org settings, including security and billing in GitHub.
- Organization members are included in the org's internal directory whereas outside collaborators are not.
- The maintain permission level allows users to manage repository settings but not be able to delete or transfer repos.
- If you want users to authenticate using a corporate identity provider set up SAML SSO.
- A GitHub Dependency graph is a representation of a repository's dependencies and dependents.
- The minimum level of support that provides help with installing and using Advanced Security is GitHub Enterprise Support.
- GitHub Support can provide assistance with GitHub account and billing queries. It can help with installing and using Advanced Security. It can helpy identify and verify the causes of suspected errors. It can help with installing and using GitHub Enterprise Server.
- You can generate and share a diagnostic file for GHES by using the *ghe-diagnostics* command-line utility to retrieve the diagnostics for your instance. You can also create a diagnostic file from the Management console by navigating to the Support tab and clicking *Download diagnostics info.*
- You can generate a support bundle in GHES by navigating to the GitHub Enterprise Server instance, selecting the *Site admin* page, then *Management Console -> Support -> Download Support Bundle.* Alternatively, you can also generate and download a support bundle directly to your local machine via SSH using the ghe-support-bundle -o > support-bundle.tgz CLI command.
- The GitHub API currently provides the following Endpoints: *Audit Log*, *Admin stats*, *License*, *Billing*, *SCIM*, *Code Security and Analysis*.
- To install a GitHub App from GitHub Marketplace for an organization you need to browse GitHub Marketplace, select the app, choose a plan, select the organization, and then review and install the app.
- The benefits and risks of using apps and actions from the GitHub Marketplace are automation of workflows and enhanced functionality, risks include potential security vulnerabilities and dependency on third-party services.
- The key implications of enabling SAML SSO for an organization in GitHub Enterprise Cloud allows organization owners to control and secure access to organization resources.

- SAML SSO for a single organization allows for different IdPs for each organization, whereas enabling it for all organizations mandates a single IdP for the entire enterprise.
- GitHub Premium Support provides SLA and written support in English 24/7.
- In order to enable and enforce SAML SSO for a single organization you need to navigate to *your organizations*, choose *settings*, click on *Authentication security*, select Enable SAML authentication, configure IdP settings, test SAML configuration, and enforce SAML SSO.
- Google Identity Platform is not officially supported and internally tested by GHEC for SAML SSO.
- You can require 2FA for an organization by going into the organization's settings, under security, select authentication security, and then choosing require two-factor authentication for everyone in your organization.
- Okta, OneLogin and Microsoft Entra ID all support GitHub Enterprise Cloud SCIM API for Organizations.
- Managed users are not allowed to contribute to public resources, and they need a separate personal account for this purpose.
- SCIM, or System for Cross-domain Identity Management, is a protocol designed to automate the identity provisioning and management. In GitHub, SCIM integrates with external Identity Providers (IdPs) to manage GHEC organization memberships, using a base URL for SCIM endpoints to perform operations like listing, inviting, and updating user identities.
- Valid Authentication methods for GitHub: OAuth tokens for third-party app integrations, SSH Key, Passkey authentication (opt-in beta for passwordless login), SAML SSO for enterprise accounts, Username and password (with optional 2FA), Personal Access Token (PAT)
- Extra features for GHEC over GitHub Free include: SAML Authentication, Ability to restrict email notifications to verified domains, Privately published GitHub Pages sites, Additional GitHub Actions minutes.
- The main restriction of EMUs is they can't interact with Public repositories.
- GitHub recommends establishing a plan for DR and configuring backups to safeguard against data loss in GHES.
- Jobs on Windows and macOS consume minutes at 2 and 10 times the rate of jobs on Linux Runners.
- The default spending limit is 0$ for GitHub Actions on monthly billed accounts, preventing additional usage beyond the included amounts.
- An Organization owner can find statistics on license usage for their GHES by navigating to the Site Admin Dashboard to view the license usage under *Enterprise account* settings.
- GitHub App managers can manage the settings of GitHub App registrations owned by the organization but do not have permissions to install or uninstall GitHub Apps.
- Within a team, a team member can either be a Team maintainer or Team member.
- git filter-repo & BFG Repo-Cleaner can be used to remove sensitive data from a Git repository's history. Before running these tools you should merge or close all open PRs.

After sensitive data is all removed, you should contact GitHub Support to remove cached views and references.

- The primary rate limit for authenticated personal users making REST API requests to GitHub API is 5,000 requests per hour.
- GitHub Apps offer granular permissions to specific repositories, while OAuth Apps request access to user data across all repositories.
- GitHub Apps subscribe to events through webhooks which notify the app of specific actions like PR openings or issue creations.
- Enterprises can track their usage of GitHub Actions by using webhooks to subscribe to information about workflow jobs and runs, and potentially using a data archiving system.
- Enterprise owners can configure IP allow lists for an enterprise on GitHub.
- When an IP allow list is enabled on your enterprise, you must only use self-hosted runners or GitHub-hosted larger runners with static IP address ranges.
- You can ensure that your self-hosted runners can communicate with GitHub when using an IP allow list by adding the IP address or IP address range of the runner to the IP allow list configured for the enterprise.