Engineer

GitHub

Microsofts hosting af git-repositories

Hvad det er

GitHub er en hostet tjeneste, der hører under Microsoft. Den opbevarer git-repositories på sine egne servere og bygger et arbejdssted rundt om dem: en webvisning af koden og dens historik, pull requests til at foreslå og gennemgå ændringer, issues til at holde styr på opgaver og GitHub Actions, der kører job automatisk, når der sker noget i et repository.

Nedenunder ligger helt almindelig git. Et repository klonet fra GitHub er et ordinært git-repository med hele sin historik, og arbejdet flytter mellem en bærbar og GitHub med de sædvanlige push og fetch. Det, GitHub lægger til, bor på tjenestens side af forbindelsen: diskussionerne i en gennemgang, issues, regler for grene, indstillinger for automatisering og listen over, hvem der må hvad, ligger i GitHubs egne systemer og ikke i det repository, man har klonet.

En pull request er GitHubs navn for et forslag om at flette ændringerne på én gren ind i en anden. Den viser forskellen linje for linje, samler kommentarer og godkendelser og viser resultatet af hvert automatisk tjek, der er kørt mod den foreslåede kode. En pull request, der er oprettet som kladde, er markeret som ufærdig og kan først flettes, når nogen markerer den klar til gennemgang.

GitHub Actions er den automatisering, der er bygget ind i tjenesten. Et workflow er en YAML-fil i repositoryets mappe `.github/workflows`. Den nævner de hændelser, der starter den, for eksempel et push, en ny pull request eller et tidspunkt, og den opremser job, der består af trin. Hvert trin kører en shell-kommando eller en action, et pakket stykke automatisering, der kan deles mellem repositories. Job kører på runners: virtuelle maskiner, som GitHub starter friske til hver kørsel med Ubuntu Linux, Windows eller macOS, eller egne maskiner, der er registreret som self-hosted runners.

To tjenester mere holder øje med, hvad der kommer ind i et repository. Dependabot læser en `dependabot.yml` -fil og opretter pull requests, der bringer afhængigheder up to date, og den opretter særskilt pull requests for afhængigheder med kendte sårbarheder. Push protection undersøger indkommende push for hemmeligheder som adgangstokens og afviser et push, der indeholder en, før det når repositoryet.

Brug begrebet i en beslutning

Brug »Hvad er GitHub?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Testet, udgivet og til at rulle tilbage, så ændringer når kunderne en tirsdag eftermiddag i stedet for sent en fredag aften.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har automatiseret GIS-SaaS-udrulning med CI/CD — udrulning, data og rapportering, alt sammen uden manuelt arbejde

Har automatiseret GIS-SaaS-udrulning med CI/CD Kontakt os →

Hvorfor det betyder noget

For mange teams er GitHub stedet, hvor arbejdet foregår, og ikke kun stedet, hvor det opbevares. Koden, diskussionen af hver ændring, optegnelsen af hvem der godkendte den, og resultatet af hvert tjek ligger side om side, så spørgsmålet om, hvorfor en linje findes, som regel kan besvares fra én side.

Automatiseringen betyder lige så meget som hostingen. Når Actions ligger ved siden af koden, kan en pull request bære sit eget bevis: testene kørte og bestod, bygget lykkedes, og scannerne rapporterede, hvad de så. Beskyttede grene gør det bevis til en regel. En gren kan sættes op til kun at modtage ændringer gennem en pull request, kun efter et bestemt antal godkendende gennemgange og kun når navngivne statustjek er bestået, så hovedlinjen kun flytter sig gennem gennemgåede og tjekkede ændringer.

Fordi GitHub sidder der, hvor koden kommer ind i et projekt, er det også her, kontrollerne af forsyningskæden bor: hvilke actions fra tredjepart et workflow må køre og i hvilken præcis version, hvordan afhængigheder holdes opdaterede, og om en hemmelighed bliver stoppet ved push. De kontroller bliver for det meste slået til og finjusteret pr. repository eller pr. organisation. En stor del af et GitHub-projekts reelle sikkerhed ligger derfor i indstillinger, som ingen ser i en kodegennemgang.

Det er byttehandlen ved en hostet tjeneste. Man får et komplet, vedligeholdt miljø uden at drive servere, og til gengæld bliver de regler, der beskytter koden, sat op i en andens brugerflade med standardvalg, der passer til det generelle tilfælde og ikke nødvendigvis til ens eget.

Sådan ser det ud i praksis

Tre sider fra virksomhedens arbejde viser GitHub i forskellige roller.

En leverancepipeline bygget på GitHub Actions deler arbejdet op i separate workflows: ét til kodekvalitetsporten, ét til testene og ét pr. miljø til deploys, med CodeQL, gennemgang af afhængigheder og et trin, der laver en softwarestykliste, ved siden af. Vejen går fra en commit til en tjeneste, der kører i Azure, og promovering flytter en build gennem miljøerne ad en fastlagt rute i stedet for i hånden. Frontenden sendes ud på et distroless-image uden shell og uden pakkehåndtering, Go-API'et på en slank Alpine-base.

Et løbende sikkerhedsprogram bruger de kontroller, der omgiver koden på GitHub. Hver GitHub Action fra tredjepart er låst til en commit-hash i stedet for et tag, så ingen kan flytte et tag og ændre, hvad et workflow kører. Dependabot holder øje med 6 økosystemer, Gitleaks scanner historikken for legitimationsoplysninger, og hver build giver en softwarestykliste med Anchore og Syft. Siden nævner også den reelle pris: mængden af fund. En scanner, der rapporterer alt, lærer folk at ignorere den, så det kræver løbende sortering at holde signalet brugbart.

Udrulningen af en GIS-applikation leveret som software-as-a-service flyttede sin cyklus med test, build og deploy over i GitHub Actions-workflows, så en udgivelse ikke længere afhang af, at nogen huskede trinene. Den tilbagevendende databehandling og rapporterne flyttede over i planlagte job i Python, Bash og SQL, og konfiguration og hemmeligheder blev standardiseret, så hvert miljø opførte sig ens.

Sådan ser en låsning ud i et workflow. Den lange hexadecimale streng er det fulde commit-id for den udgivelse, der blev gennemgået, og kommentaren holder den læsbare version ved siden af:

```
steps:
    - uses: actions/checkout@3d3c42e5aac5ba805825da76410c181273ba90b1 # v7.0.1
```

Hvor det går galt

- **En action låst til et tag ændrer sig uden varsel.** Et tag som `v4` kan flyttes eller slettes af den, der styrer actionens repository, så den kode, et workflow kører i morgen, kan være en anden end den, der blev gennemgået i dag. GitHubs egen anbefaling er at låse til en fuld commit-SHA, som en angriber kun kan efterligne ved at fremstille en SHA-1-kollision for et gyldigt git-objekt.
- **Et workflow-token har mere magt, end jobbet behøver.** Hver kørsel får et `GITHUB_TOKEN`, og dets rettigheder afgør, hvad et kompromitteret trin kan ændre. GitHub anbefaler at gøre læseadgang til repositoryets indhold til standard og kun give skriveadgang til de job, der har brug for den.
- **Utroværdig tekst bliver kørt som en kommando.** Et workflow, der skriver titlen på en pull request direkte ind i et inline-script, lader den, der skrev titlen, bestemme en del af scriptet. Sættes værdien i stedet som en mellemliggende miljøvariabel, forbliver den data. Udløserne `pull_request_target` og `workflow_run` starter privilegerede workflows, og GitHub advarer om, at det udsætter repositoryet for kompromittering at tjekke en utroværdig pull request ud i et af dem.
- **Administratorer går lige forbi reglerne for grene.** Som standard gælder begrænsningerne i en regel for beskyttede grene ikke for personer med administratorrettigheder til repositoryet. Slås "Do not allow bypassing the above settings" til, binder reglerne også dem.
- **En hemmelighed slipper igennem push protection.** Den form, der er slået til som standard, beskytter hver brugers egne push til offentlige repositories. Den form, der beskytter et repository mod alle bidragydere, kræver GitHub Secret Protection og er slået fra, indtil nogen slår den til, og selv da kan alle med skriveadgang omgå en blokering ved at angive en begrundelse, hvilket udløser en advarsel og en post i revisionsloggen. Når en nøgle først er skubbet, findes den i hver klon, så skift den i stedet for at stole på en omskrivning af historikken.

Læs mere

- GitHub Docs — hele dokumentationen for repositories, pull requests, Actions og sikkerhedsfunktionerne.
- Understanding GitHub Actions — workflows, hændelser, job, trin og runners, og hvordan de hænger sammen.
- Secure use reference — låsning af actions, tokenrettigheder og scriptinjektion, beskrevet af GitHub selv.
- About protected branches — krævede gennemgange, krævede statustjek, og hvem der kan omgå dem.

- Dependabot version updates — hvordan en `dependabot.yml` -fil styrer opdateringernes pull requests.
- Push protection — de to former for push protection, hvad hver af dem blokerer, og hvordan en omgåelse bliver registreret.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er GitHub?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/github/