Engineer

GitHub

Microsofts vertstjeneste for git-repoer

Hva det er

GitHub er en vertstjeneste som hører til Microsoft. Tjenesten oppbevarer git-repoer på sine egne servere og bygger en arbeidsplass rundt dem: en webvisning av koden og historikken, pull requests for å foreslå og gjennomgå endringer, saker for å følge opp arbeid, og GitHub Actions for å kjøre jobber automatisk hver gang noe skjer i et repo.

Under ligger helt vanlig git. Et repo som er klonet fra GitHub, er et ordinært git-repo med hele historikken, og arbeid flyttes mellom en bærbar maskin og GitHub med de vanlige kommandoene for push og fetch. Det GitHub legger til, ligger på tjenestens side av forbindelsen: diskusjoner fra gjennomganger, saker, regler for grener, innstillinger for automatisering og listen over hvem som får gjøre hva, oppbevares i GitHubs egne systemer og ikke i repoet du klonet.

En pull request er GitHubs navn på et forslag om å flette endringene på én gren inn i en annen. Den viser forskjellen linje for linje, samler kommentarer og godkjenninger og viser resultatet av hver automatiske kontroll som har kjørt mot den foreslåtte koden. En pull request som åpnes som utkast, er merket som uferdig, og den kan først flettes når noen har merket den som klar for gjennomgang.

GitHub Actions er automatiseringen som er bygd inn i tjenesten. En arbeidsflyt er en YAML-fil i katalogen `.github/workflows` i repoet. Den navngir hendelsene som starter den, for eksempel en push, en ny pull request eller en tidsplan, og lister opp jobber som består av steg. Hvert steg kjører en skallkommando eller en action, en pakket bit automatisering som kan deles mellom repoer. Jobbene kjører på runnere: virtuelle maskiner som GitHub starter friske for hver kjøring, med Ubuntu Linux, Windows eller macOS, eller egne maskiner som er registrert som selvhostede runnere.

To tjenester til holder øye med det som kommer inn i et repo. Dependabot leser en `dependabot.yml` -fil og åpner pull requests som oppdaterer avhengigheter, og den åpner i tillegg egne pull requests for avhengigheter med kjente sårbarheter. Push-beskyttelse undersøker innkommende push etter hemmeligheter som tilgangstokener, og blokkerer en push som inneholder en, før den når repoet.

Bruk begrepet i en beslutning

Bruk «Hva er GitHub?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

For mange team er GitHub stedet der arbeidet skjer, og ikke bare der det lagres. Koden, diskusjonen om hver endring, oversikten over hvem som godkjente den, og utfallet av hver kontroll ligger side om side, så spørsmålet om hvorfor en linje finnes, kan som regel besvares fra én side.

Automatiseringen betyr like mye som selve vertstjenesten. Med Actions ved siden av koden kan en pull request bære sine egne bevis: testene kjørte og besto, bygget gikk gjennom, skannerne rapporterte det de så. Beskyttede grener gjør disse bevisene til en regel. En gren kan settes opp til å ta imot endringer bare gjennom en pull request, bare etter et fastsatt antall godkjennende gjennomganger og bare når navngitte statuskontroller har bestått. Da beveger hovedlinjen seg utelukkende gjennom endringer som er gjennomgått og kontrollert.

Fordi GitHub står akkurat der koden kommer inn i et prosjekt, er det også der kontrollene for forsyningskjeden bor: hvilke tredjeparts actions en arbeidsflyt får kjøre og i hvilken eksakt versjon, hvordan avhengighetene holdes oppdatert, og om en hemmelighet blir stoppet ved pushen. Disse kontrollene slås stort sett på og finjusteres per repo eller per organisasjon. En stor del av den reelle sikkerheten i et GitHub-prosjekt ligger derfor i innstillinger, og dem ser ingen i en kodegjennomgang.

Det er avveiningen med en vertstjeneste. Dere får et komplett, vedlikeholdt miljø uten å drifte servere, og til gjengjeld blir reglene som beskytter koden deres, konfigurert i noen andres grensesnitt, med standardverdier valgt for det generelle tilfellet og ikke for deres.

Slik ser det ut i praksis

Tre sider fra selskapets arbeid viser GitHub i ulike roller.

En leveransepipeline bygd på GitHub Actions deler arbeidet i separate arbeidsflyter: én for kvalitetsporten for koden, én for testene og én per miljø for produksjonssetting, med CodeQL, avhengighetsgjennomgang og et steg som lager en stykkliste for programvaren, ved siden av. Veien går fra en commit til en tjeneste som kjører i Azure, og promotering flytter et bygg gjennom miljøene langs en fast rute i stedet for for hånd. Frontenden leveres på et distroless-image uten skall og uten pakkebehandler, Go-API-et på en slank Alpine-base.

Et kontinuerlig sikkerhetsprogram bruker kontrollene som omgir koden på GitHub. Hver tredjeparts GitHub Action er låst til en commit-hash i stedet for en tagg, slik at ingen kan flytte en tagg og endre hva en arbeidsflyt kjører. Dependabot overvåker 6 økosystemer, Gitleaks skanner historikken etter påloggingsinformasjon, og hvert bygg lager en stykkliste for programvaren med Anchore og Syft. Siden nevner også den egentlige kostnaden: mengden funn. En skanner som rapporterer alt, lærer folk å overse den, så det krever løpende triagering å holde signalet brukbart.

Produksjonssettingen av en SaaS-applikasjon for GIS flyttet syklusen med test, bygg og produksjonssetting inn i arbeidsflyter i GitHub Actions, slik at en utgivelse sluttet å avhenge av at noen husket stegene. Den gjentakende databehandlingen og rapportene ble flyttet til planlagte jobber i Python, Bash og SQL, og konfigurasjonen og hemmelighetene ble standardisert slik at alle miljøer oppførte seg likt.

Låsing ser slik ut i en arbeidsflyt. Den lange heksadesimale strengen er den fullstendige commit-identifikatoren til utgivelsen som ble gjennomgått, og kommentaren holder den lesbare versjonen ved siden av:

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

Hvor det går galt

- **En action som er låst til en tagg, endrer seg uten forvarsel.** En tagg som `v4` kan flyttes eller slettes av den som kontrollerer repoet til actionen, så koden en arbeidsflyt kjører i morgen, kan være en annen enn koden som ble gjennomgått i dag. GitHubs egen veiledning er å låse til en full commit-SHA, som en angriper bare kunne etterlignet ved å lage en SHA-1-kollisjon for et gyldig git-objekt.
- **Et token i arbeidsflyten har mer makt enn jobben trenger.** Hver kjøring får et `GITHUB_TOKEN` , og rettighetene til det avgjør hva et kompromittert steg kunne endret. GitHub anbefaler å gjøre lesetilgang til innholdet i repoet til standard og å gi skrivetilgang bare til jobbene som trenger det.
- **Tekst utenfra kjøres som en kommando.** En arbeidsflyt som skriver tittelen på en pull request rett inn i et innebygd skript, lar den som skrev tittelen, bestemme en del av skriptet. Settes verdien i en mellomliggende miljøvariabel, forblir den data. Utløserne `pull_request_target` og `workflow_run` starter arbeidsflyter med utvidede rettigheter, og GitHub advarer om at det å sjekke ut en pull request man ikke stoler på, inne i en slik arbeidsflyt åpner repoet for kompromittering.
- **Administratorer går rett forbi grenreglene.** Som standard gjelder begrensningene i en regel for grenbeskyttelse ikke for folk med administratorrettigheter i repoet. Slår du på «Do not allow bypassing the above settings», binder reglene dem også.
- **En hemmelighet slipper gjennom push-beskyttelsen.** Varianten som er på som standard, beskytter hver brukers egne push til offentlige repoer. Varianten som vokter et repo mot alle bidragsytere, krever GitHub Secret Protection og står avslått til noen slår den på. Selv da kan alle med skrivetilgang omgå en blokkering ved å oppgi en grunn, noe som utløser et varsel og havner i revisjonsloggen. Når påloggingsinformasjon først er pushet, ligger den i hver klone, så bytt den ut i stedet for å stole på at en omskriving av historikken redder deg.

Les mer

- GitHub Docs — den fullstendige dokumentasjonen for repoer, pull requests, Actions og sikkerhetsfunksjonene.
- Understanding GitHub Actions — arbeidsflyter, hendelser, jobber, steg og runnere, og hvordan de henger sammen.
- Secure use reference — låsing av actions, rettigheter for tokener og skriptinjeksjon, beskrevet av GitHub selv.
- About protected branches — påkrevde gjennomganger, påkrevde statuskontroller og hvem som kan omgå dem.

- Dependabot version updates — hvordan en `dependabot.yml` -fil styrer pull requestene med oppdateringer.
- Push protection — de to typene push-beskyttelse, hva hver av dem blokkerer, og hvordan en omgåelse blir registrert.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er GitHub?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/github/