Engineer

GitHub

värdtjänsten från Microsoft för git-repon

Vad det är

GitHub är en värdtjänst, en del av Microsoft, som håller git-repon på sina egna servrar och bygger en arbetsplats runt dem: en webbvy av koden och dess historik, pull requests för att föreslå och granska ändringar, ärenden (issues) för att hålla reda på arbetet, och GitHub Actions för att köra jobb automatiskt så fort något händer i ett repo.

Därunder finns vanligt git. Ett repo som klonas från GitHub är ett helt vanligt git-repo med hela sin historik, och arbetet flyttas mellan en laptop och GitHub med de vanliga kommandona för push och fetch. Det som GitHub lägger till lever på tjänstens sida av förbindelsen: granskningsdiskussioner, ärenden, grenregler, inställningar för automatiseringen och listan över vem som får göra vad hålls i GitHubs egna system och inte i det repo du klonade.

En pull request är GitHubs namn på ett förslag om att föra in ändringarna på en gren i en annan gren. Den visar skillnaden rad för rad, samlar kommentarer och godkännanden och visar resultatet av varje automatisk kontroll som har körts mot den föreslagna koden. En pull request som öppnas som utkast är markerad som ofärdig, och den går inte att slå ihop förrän någon markerar den som klar för granskning.

GitHub Actions är den automatisering som är inbyggd i tjänsten. Ett arbetsflöde är en YAML-fil i repots katalog `.github/workflows`. Filen namnger de händelser som startar arbetsflödet, t.ex. en push, en ny pull request eller ett schema, och listar jobb som består av steg, där varje steg kör ett skalkommando eller en action, en paketerad bit automatisering som kan delas mellan repon. Jobben körs på runners: virtuella maskiner som GitHub startar på nytt för varje körning med Ubuntu Linux, Windows eller macOS, eller maskiner som du själv driver och har registrerat som self-hosted runners.

Två tjänster till bevakar det som kommer in i ett repo. Dependabot läser en fil `dependabot.yml` och öppnar pull requests som uppdaterar beroenden, och separat skapar den pull requests för beroenden med kända sårbarheter. Push protection granskar inkommande pushar efter hemligheter som åtkomsttokens och stoppar en push som innehåller en sådan innan den når repot.

Använd begreppet i ett beslut

Använd ”Vad är GitHub?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

För många team är GitHub platsen där arbetet görs, inte bara där det lagras. Koden, diskussionen om varje ändring, uppgiften om vem som godkände den och utfallet av varje kontroll ligger sida vid sida, så frågan om varför en rad finns kan oftast besvaras från en enda sida.

Automatiseringen betyder lika mycket som värdskapet. När Actions ligger bredvid koden kan en pull request bära sina egna bevis: testerna kördes och gick igenom, bygget fungerade, skannrarna rapporterade vad de såg. Skyddade grenar gör de bevisen till en regel. En gren kan ställas in så att den bara tar emot ändringar via en pull request, bara efter ett bestämt antal godkännande granskningar och bara när namngivna statuskontroller har gått igenom, så att huvudlinjen förs framåt av granskade och kontrollerade ändringar och inget annat.

Eftersom GitHub sitter där koden kommer in i ett projekt är det också där kontrollerna för mjukvarans leverantörskedja finns: vilka actions från tredje part ett arbetsflöde får köra och i exakt vilken version, hur beroenden hålls aktuella och om en hemlighet stoppas redan vid pushen. De här kontrollerna slås för det mesta på och finjusteras per repo eller per organisation. En stor del av den verkliga säkerheten i ett GitHub-projekt finns därför i inställningar, och dem ser ingen vid en kodgranskning.

Det är kompromissen med en värdtjänst. Du får en komplett och underhållen miljö utan att behöva driva servrar, och i gengäld ställs reglerna som skyddar din kod in i någon annans gränssnitt, med standardvärden som är valda för det allmänna fallet och inte för ditt.

Så ser det ut i praktiken

Tre sidor från företagets arbete visar GitHub i olika roller.

En leveranspipeline byggd på GitHub Actions delar upp arbetet i separata arbetsflöden: ett för kvalitetsgrinden för koden, ett för testerna och ett per miljö för driftsättningar, med CodeQL, beroendegranskning och ett steg som tar fram en materialförteckning för mjukvaran parallellt med dem. Vägen går från en commit till en tjänst som körs i Azure, och ett bygge förs vidare genom miljöerna längs en fastställd väg i stället för manuellt. Frontenden levereras på en distroless-avbildning som saknar både skal och pakethanterare, Go-API:et på en slimmad Alpine-bas.

Ett kontinuerligt säkerhetsprogram använder de kontroller som omger koden på GitHub. Varje GitHub Action från tredje part är låst till en commit-hash i stället för en tagg, så att ingen kan flytta en tagg och därmed ändra vad ett arbetsflöde kör. Dependabot bevakar 6 ekosystem, Gitleaks söker igenom historiken efter inloggningsuppgifter, och varje bygge tar fram en materialförteckning för mjukvaran med Anchore och Syft. Sidan nämner också den verkliga kostnaden: mängden fynd. En skanner som rapporterar allt lär folk att ignorera den, så att hålla signalen användbar kräver löpande triagering.

Driftsättningen av en GIS-applikation som levereras som tjänst flyttade sin cykel med test, bygge och driftsättning in i pipelines i GitHub Actions, så att en release slutade hänga på att någon kom ihåg stegen. Den återkommande databehandlingen och rapporterna flyttades in i schemalagda jobb i Python, Bash och SQL, och konfigurationen och hemligheterna standardiserades så att varje miljö betedde sig likadant.

Så här ser en låsning ut i ett arbetsflöde. Den långa hexadecimala strängen är den fullständiga commit-identifieraren för den release som granskades, och kommentaren håller den läsbara versionen bredvid:

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

Var det går fel

- **En action som är låst till en tagg ändras utan förvarning.** En tagg som `v4` kan flyttas eller raderas av den som kontrollerar actionens repo, så koden som ett arbetsflöde kör i morgon kan skilja sig från koden som granskades i dag. GitHubs egen vägledning är att låsa till en fullständig commit-SHA, som en angripare bara kan efterlikna genom att ta fram en SHA-1-kollision för ett giltigt git-objekt.
- **Arbetsflödets token har mer makt än jobbet behöver.** Varje körning får en `GITHUB_TOKEN`, och dess behörigheter avgör vad ett kompromitterat steg skulle kunna ändra. GitHub rekommenderar att läsåtkomst till repots innehåll är standard och att skrivåtkomst bara ges till de jobb som behöver den.
- **Text som vem som helst kan skriva körs som ett kommando.** Ett arbetsflöde som skriver in titeln på en pull request direkt i ett inbäddat skript låter den som skrev titeln välja en del av skriptet. Att sätta värdet som en mellanliggande miljövariabel gör att det förblir data. Utlösarna `pull_request_target` och `workflow_run` startar privilegierade arbetsflöden, och GitHub varnar för att det utsätter repot för intrång att checka ut en obetrodd pull request i ett sådant arbetsflöde.
- **Administratörer går rakt förbi grenreglerna.** Som standard gäller begränsningarna i en regel för grenskydd inte för personer med administratörsbehörighet i repot. Att slå på ”Do not allow bypassing the above settings” gör att reglerna binder även dem.
- **En hemlighet slinker igenom push protection.** Den form som är påslagen som standard skyddar varje användares egna pushar till publika repon. Den form som skyddar ett repo mot varje bidragsgivare kräver GitHub Secret Protection och är avslagen tills någon slår på den, och även då kan alla med skrivåtkomst kringgå en blockering genom att ange ett skäl, vilket utlöser ett larm och hamnar i granskningsloggen. När en åtkomstuppgift väl har pushats finns den i varje klon, så byt ut den i stället för att lita på en omskrivning av historiken.

Läs mer

- GitHub Docs — hela dokumentationen för repon, pull requests, Actions och säkerhetsfunktionerna.
- Understanding GitHub Actions — arbetsflöden, händelser, jobb, steg och runners, och hur de hänger ihop.

- Secure use reference — låsning av actions, behörigheter för token och skriptinjektion, beskrivet av GitHub självt.
- About protected branches — obligatoriska granskningar, obligatoriska statuskontroller och vem som får kringgå dem.
- Dependabot version updates — hur en fil `dependabot.yml` styr de pull requests som uppdaterar beroenden.
- Push protection — de två formerna av push protection, vad var och en blockerar och hur ett kringgående registreras.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är GitHub?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/github/