Engineer

Ansible

et verktøy som setter en server i en nedskrevet tilstand

Hva det er

Ansible er automatiseringsprogramvare med åpen kildekode som kobler seg til maskiner over SSH og bringer hver av dem til konfigurasjonen som er beskrevet i et sett YAML-filer: hvilke programvarepakker som er installert, hvilke brukere og filer som finnes, hvilke tjenester som kjører, og hvordan de er satt opp. Maskinen Ansible kjøres fra, heter kontrollnoden (control node), og maskinene den konfigurerer, er administrerte noder (managed nodes). På dem trenger du ingen agent, fordi Ansible logger inn med operativsystemets egne påloggingsdetaljer og gjør jobben over den forbindelsen. Filene beskriver resultatet, og Ansible finner ut på hver vert om det resultatet allerede er på plass.

Delene har enkle navn. Et inventar (inventory) lister opp de administrerte nodene og sorterer dem i grupper som `web` eller `db`, med variabler som gjelder en hel gruppe eller én enkelt vert. En playbook inneholder ett eller flere play, og hvert play kobler en gruppe verter til en liste med oppgaver (tasks). Hver oppgave kaller en modul, et lite program som vet hvordan én type ting skal forvaltes: en programvarepakke, en fil, en tjeneste, en bruker, en brannmurregel. Prosjektets egen dokumentasjon beskriver playbooks som et gjentakbart, gjenbrukbart og enkelt system for konfigurasjonsstyring og utrulling til mange maskiner. En rolle samler oppgaver, maler, standardvariabler og handlers under ett navn, så «en PostgreSQL-server» eller «en herdet SSH-daemon» skrives én gang og brukes på hver vert som trenger den.

Et kort play viser formen:

```
- name: Web servers serve the site
  hosts: web
  become: true
  tasks:
    - name: nginx is installed
```

```
    ansible.builtin.package:
      name: nginx
      state: present

  - name: nginx runs now and after a reboot
    ansible.builtin.service:
      name: nginx
      state: started
      enabled: true
```

Hver oppgave navngir en tilstand og ikke en handling. De fleste moduler sjekker først om tilstanden allerede gjelder, og avslutter uten å gjøre noe hvis den gjør det, så den andre kjøringen av dette playet på en ferdig konfigurert vert endrer ingenting og sier det. Den egenskapen, at oppgaven er idempotent, er det som gjør det trygt å kjøre den samme playbooken mot hele flåten hver gang, i stedet for å holde rede på hvilken vert som har fått hvilket skript.

Folk som spør om Ansible, veier det ofte mot tre naboer. Mot et skallskript er forskjellen at skriptet lister opp steg og kjører dem på nytt enten de trengs eller ikke, mens en modul sjekker før den handler. Mot Terraform eller Bicep er forskjellen laget: de verktøyene oppretter ressurser i skyen, som nettverk, databaser og virtuelle maskiner, mens Ansible konfigurerer det som kjører inne i en maskin når den først finnes. Mange team bruker ett av hver. Mot Puppet og Chef er forskjellen modellen: de kjører som regel en agent på hver maskin som henter konfigurasjonen sin fra en sentral server, mens Ansible dytter ut endringene fra der det blir startet, og trenger bare SSH i den andre enden.

Bruk begrepet i en beslutning

Bruk «Hva er Ansible?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Hele oppsettet skrevet ned som filer, slik at det kan gjenskapes nøyaktig og aldri bare finnes i hodet på én person.

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. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

En server som er satt opp for hånd, er bare så reproduserbar som hukommelsen til den som satte den opp. Et halvt år senere kan ingen si sikkert hvorfor en parameter i systemkjernen har verdien sin, hvilken pakke som ble låst til en

bestemt versjon og hvorfor, eller hva som må skje først når maskinen bygges på nytt. Skriver du oppsettet ned som playbooks, blir den hukommelsen til filer som kan leses, gjennomgås i en pull request og kjøres på nytt på en fersk maskin med samme resultat.

Den skrevne formen endrer også hva avvik (drift) betyr. Når playbooken er beskrivelsen av verten, rapporterer en kjøring i check mode mot en kjørende maskin hvert sted der de to er uenige. Med flagget `--diff` viser rapporten linjene i hver fil som ville blitt endret. En flåte som konvergeres jevnlig, holder seg nær beskrivelsen sin, og en vert som er redigert for hånd, dukker opp som en endring neste gang playbooken kjøres.

Modellen har sine kostnader. Ansible handler bare når det kjøres, så avvik mellom to kjøringer blir stående til den neste. Oppgavene kjører i rekkefølge, én om gangen på tvers av alle verter som matcher, og en stor playbook mot mange maskiner er tregere enn et skript som gjør én ting. Logikk skrevet i YAML og Jinja2-maler blir vanskelig å følge når en rolle samler mange betingelser. Og en kjøring som stopper halvveis, etterlater vertene sine delvis endret, fordi Ansible ikke har noen transaksjon å rulle tilbake.

Slik ser det ut i praksis

Flere prosjekter på dette nettstedet viser hvordan et nedskrevet driftsmiljø ser ut over tid, og hva som skal til for å holde det ærlig.

Selskapets infrastruktur som kode dekker offentlige tjenester i drift og konfigurasjon for andre komponenter, blant annet backup- og e-postautomatisering som aktiveres i et eget trinn. Den registrerte implementasjonen omfatter 19 playbooks og 34 roller fordelt på 12.065 linjer YAML; tallene beskriver det daværende stadiet, ikke repositoriets nåværende størrelse. En vert tilhører en nivågruppe der variablene angir hvilke roller den kjører, så provisjonering uten argumenter bringer hver vert til sin deklarerte tilstand. Idempotens behandles som en kontrakt: en konvergert vert rapporterer null endringer, og en play som ikke kan si det, er ikke ferdig. Fire kommandonavnerom holder løftene adskilt, fra en kvalitetsport som ikke berører noen vert, til verifikasjon som endrer en vert med hensikt og avlegger en vurdering. Siden er tydelig om kostnaden: hver endring tar lengre tid enn å redigere en fil direkte på serveren, og en konvergering som gjennomføres halvveis, er verre enn en som nekter å starte.

Tilgangskontroll er et annet sted der en skriftlig beskrivelse lønner seg. 1.000 RBAC-regler ble konfigurert med Ansible på tvers av utviklermiljøer, applikasjonsinstanser, PostgreSQL, MS SQL og Linux-servere, med hver regel knyttet til én rolle og ett system. Det registrerte resultatet er et dokumentert rollebasert regelsett og en repeterbar måte å anvende og gjennomgå det på. Det er dette registreringen underbygger; den tallfester ikke redusert sikkerhetsrisiko eller spart oppsettstid.

Å holde tørrkjøringen sannferdig krevde et eget arbeid. Check mode ble gjort ærlig på hele plattformen etter at en tørrkjøring i september 2026 stoppet på en assert som tok feil om verten, og rådet operatøren til å sette et flagg som ville ha lempet på en sikkerhetssandkasse. En gjennomgang fant seks prober som avgjorde ut fra verdier verten aldri hadde gitt. Prober som leser tilstand kjøringen ikke har rørt, kjører nå også i check mode. Resten hoppes over med en melding som navngir hva som gikk uverifisert, og regelen gikk inn i utviklingsveiledningen.

Hvor det går galt

Det meste av trøbbelet med Ansible kommer av å stole mer på en rapport enn på mekanismen bak den.

En tørrkjøring melder et rent resultat for kommandoer den aldri kjørte. Check mode er en simulering. Moduler som støtter den, forutsier endringen sin, og ifølge dokumentasjonen rapporterer moduler som mangler støtte, ingenting og gjør ingenting. Modulene `command` og `shell` støtter den bare delvis: med mindre `creates` eller `removes` er angitt, hopper de over kommandoen og registrerer et resultat med tom utdata. En senere `when:` som leser det resultatet, avgjør ut fra en verdi ingen har målt. En probe som bare leser verten, kan få beskjed om å kjøre likevel, og en probe som ikke kan kjøre, bør si det:

```
- name: Read the firewall state, even in check mode
  ansible.builtin.command: ufw status
  register: fw
  check_mode: false
  changed_when: false

- name: Say what this dry run did not verify
  ansible.builtin.debug:
    msg: "check mode: the service restart was not exercised"
  when: ansible_check_mode
```

En kjøring stopper på den niende oppgaven med de åtte første allerede utført. Når en oppgave feiler på en vert, tar Ansible den verten ut av resten av kjøringen og lar det som allerede er endret, bli stående. Sjekker som kan kjøres før noe berører en vert, hører hjemme foran konvergeringen. Det var akkurat det som skjedde etter en endring av driftshemmelighetene i driftsmiljøet bak dette nettstedet: et preflight-play validerer nå vaulten lokalt først, med de samme oppgavefilene som den virkelige konvergeringen og på omtrent et sekund. En hemmelighet som ikke lar seg slå opp, koster dermed en nektelse på to sekunder på en bærbar maskin i stedet for en halvveis gjennomført endring på en kjørende server. Den samme siden peker på en felle: facts som settes i løpet av et play, lever lenger enn playet, så en sjekk og en konvergering som er lenket sammen i én kommando, kan dele en verdi sjekken burde ha lest på nytt.

En vert ingen har rørt, melder endringer ved hver kjøring. Modulene `command` og `shell` melder en endring hver gang de kjører, fordi de ikke kan vite hva kommandoen faktisk gjorde. En playbook full av dem konvergerer aldri til null, og ekte endringer drukner i støyen. Velg heller en modul som forvalter tingen direkte, legg til `creates` eller `removes`, eller fortell Ansible med `changed_when` når utdata betyr en endring.

Et passord dukker opp i en logg eller i repoet. Variabler er ren tekst til noe krypterer dem. Ansible Vault krypterer variabelfiler eller enkeltverdier så de kan ligge i repoet, og `no_log: true` på en oppgave holder argumentene og resultatene ute av utdataene. Kjør playbooken én gang med høy detaljgrad i et testmiljø, og les hva den skriver ut, før du stoler på noen av delene.

Les mer

- Ansible Community Documentation — prosjektets egen inngang til installasjon, begreper og hver eneste veiledning.
- Ansible playbooks — play, oppgaver, moduler, kjørerekkefølge og hva det betyr i praksis at noe er idempotent.
- Validating tasks: check mode and diff mode — hva en tørrkjøring kan og ikke kan se, og bryterne som finnes per oppgave.
- How to build your inventory — verter, grupper og variablene som hører til dem.
- Protecting sensitive data with Ansible Vault — kryptering av filer og enkeltverdier som ligger ved siden av playbookene.
- ansible.builtin.command — modulreferansen, med den delvise støtten for check mode.

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 Ansible?» 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/ansible/