Engineer

Ansible

ett verktyg som för en server till ett nedskrivet tillstånd

Vad det är

Ansible är automationsprogramvara med öppen källkod som ansluter till maskiner över SSH och för var och en till den konfiguration som beskrivs i en uppsättning YAML-filer: vilka programpaket som är installerade, vilka användare och filer som finns, vilka tjänster som körs och hur de är inställda. Maskinen som Ansible körs från kallas kontrollnod, och maskinerna den konfigurerar är hanterade noder. På dem behöver ingen agent installeras, eftersom Ansible loggar in med operativsystemets egna inloggningsuppgifter och gör sitt arbete över den anslutningen. Filerna anger resultatet, och Ansible kontrollerar på varje värddator om det resultatet redan gäller.

Delarna har enkla namn. Ett inventory listar de hanterade noderna och delar in dem i grupper som `web` eller `db`, med variabler som gäller för en hel grupp eller för en enskild värd. En playbook rymmer en eller flera plays, och varje play kopplar en grupp värdar till en lista med uppgifter (tasks). Varje uppgift anropar en modul, ett litet program som vet hur en sorts sak hanteras: ett paket, en fil, en tjänst, en användare, en regel i en brandvägg. Projektets egen dokumentation beskriver playbooks som ett upprepbart, återanvändbart och enkelt system för konfigurationshantering och driftsättning på flera maskiner. En roll samlar uppgifter, mallar, standardvariabler och handlers under ett namn, så att ”en PostgreSQL-server” eller ”en härdad SSH-daemon” skrivs en gång och tillämpas på varje värd som behöver den.

En kort play visar 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
```

Varje uppgift anger ett tillstånd, inte en handling. De flesta moduler kontrollerar först om tillståndet redan gäller och avslutar utan att göra något om det gör det, så en andra körning av den här playen på en konfigurerad värd ändrar ingenting och säger det. Den egenskapen, idempotens, är det som gör det säkert att köra samma playbook mot hela flottan varje gång, i stället för att hålla reda på vilken värd som har fått vilket skript.

Den som frågar om Ansible jämför det ofta med tre grannar. Mot ett skalskript är skillnaden att ett skript räknar upp steg och kör dem igen oavsett om de behövs, medan en modul kontrollerar innan den agerar. Mot Terraform eller Bicep är skillnaden skiktet: de verktygen skapar resurser i ett moln, som nätverk, databaser och virtuella maskiner, medan Ansible konfigurerar det som körs inuti en maskin när den väl finns. Många team använder ett av varje. Mot Puppet och Chef är skillnaden modellen: de kör oftast en agent på varje maskin som hämtar sin konfiguration från en central server, medan Ansible skjuter ut ändringar från den plats där det startas och bara behöver SSH på andra sidan.

Använd begreppet i ett beslut

Använd ”Vad är Ansible?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Hela uppsättningen nedskriven som filer, så att den kan återskapas exakt och aldrig bara finns i en enda persons huvud.

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. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

En server som har satts upp för hand går bara att återskapa så länge den som satte upp den minns hur. Ett halvår senare kan ingen med säkerhet säga varför en parameter i systemkärnan har sitt värde, vilket paket som låstes vid en version

och varför, eller vad som måste hända först när maskinen byggs om. När uppsättningen skrivs ned som playbooks blir minnet filer som kan läsas, granskas i en pull request och köras igen på en ny maskin med samma resultat.

Den skrivna formen ändrar också vad avdrift betyder. När playbooken är beskrivningen av värden rapporterar en körning i check mode mot en körande maskin varje ställe där de två inte stämmer överens. Med flaggan `--diff` visar rapporten de rader i varje fil som skulle ändras. En flotta som konvergeras regelbundet håller sig nära sin beskrivning, och en värd som har redigerats för hand syns som en ändring nästa gång playbooken körs.

Modellen har sina kostnader. Ansible agerar bara när det körs, så avdrift mellan körningarna ligger kvar till nästa gång. Uppgifterna körs i ordning, en i taget över alla matchade värdar, och en stor playbook mot många maskiner är långsammare än ett skript som gör en enda sak. Logik som skrivs i YAML och Jinja2-mallar blir svår att följa när en roll väl har fått många villkor. Och en körning som stannar halvvägs lämnar sina värdar delvis ändrade, eftersom Ansible saknar en transaktion att rulla tillbaka.

Så ser det ut i praktiken

Flera projekt på den här webbplatsen visar hur en nedskriven IT-miljö ser ut över tid, och vad som krävs för att hålla den ärlig.

Företagets infrastruktur som kod omfattar offentliga tjänster i drift och konfiguration för andra komponenter, däribland backup- och e-postautomatisering vars aktivering är ett separat steg. Den dokumenterade implementeringen omfattar 19 playbooks och 34 roller fördelade på 12.065 rader YAML; siffrorna beskriver det dåvarande skedet, inte kodförrådets nuvarande storlek. En värd tillhör en nivågrupp vars variabler anger vilka roller den kör, så provisionering utan argument för varje värd till dess deklarerade tillstånd. Idempotens behandlas som ett kontrakt: en konvergerad värd rapporterar noll ändringar, och en play som inte kan säga det är inte färdig. Fyra kommandonamnrymder håller löftena åtskilda, från en kvalitetsgrind som inte rör någon värd till verifiering som avsiktligt ändrar en värd och lämnar ett utslag. Sidan är tydlig med kostnaden: varje ändring tar längre tid än att redigera en fil direkt på servern, och en konvergering som genomförs till hälften är sämre än en som vägrar börja.

Åtkomstkontroll är ett annat område där en skriftlig beskrivning lönar sig. 1.000 RBAC-regler konfigurerades med Ansible i utvecklingsmiljöer, applikationsinstanser, PostgreSQL, MS SQL och Linux-servrar, med varje regel knuten till en roll och ett system. Det dokumenterade resultatet är en beskriven rollbaserad regeluppsättning och ett repeterbart sätt att tillämpa och granska den. Det är detta underlaget styrker; det kvantifierar inte minskad säkerhetsrisk eller sparad tid för konfigurering.

Att hålla torrkörningen sanningsenlig krävde ett eget arbete. Check mode fick tala sanning i hela plattformen efter att en torrkörning i september 2026 föll på en kontroll som hade fel om värden och rådde operatören att sätta en flagga som skulle ha luckrat upp en säkerhetssandlåda. En genomsökning hittade sex sonder som fattade beslut utifrån värden som maskinen aldrig hade gett. Sonder som läser tillstånd som körningen inte har rört körs nu även i check mode. Resten hoppas över med ett meddelande som anger vad som inte verifierades, och regeln fördes in i utvecklarguiden.

Var det går fel

De flesta problem med Ansible kommer av att man litar mer på en rapport än på mekanismen bakom den.

En torrkörning rapporterar ett rent resultat för kommandon som den aldrig körde. Check mode är en simulering. Moduler som stöder läget förutsäger sin ändring, och enligt dokumentationen rapporterar moduler utan stöd ingenting och gör ingenting. Modulerna `command` och `shell` stöder läget bara delvis: om `creates` eller `removes` saknas hoppar de över kommandot och registrerar ett resultat med tom utdata. Varje senare `when:` som läser det resultatet fattar beslut utifrån ett värde som ingen har mätt. En sond som bara läser värden kan få order att köras ändå, och en som inte kan köras bör säga 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 körning stannar vid sin nionde uppgift med de åtta första redan tillämpade. När en uppgift misslyckas på en värd släpper Ansible den värden från resten av körningen och lämnar det som redan har ändrats kvar. Kontroller som kan köras innan något rör en värd hör hemma före konvergeringen. Så gick det efter en ändring av hemligheterna i företagets egen IT-miljö, och sedan dess validerar en play för förhandskontroll vault-filen lokalt först, med samma task-filer som den riktiga konvergeringen och på ungefär en sekund, så en hemlighet som inte går att slå upp kostar en vägran på två sekunder på en bärbar dator i stället för en delvis tillämpad ändring på en körande server. Samma sida pekar ut en fälla: fakta som sätts under en play överlever den, så en kontroll och en konvergering som kedjas ihop i ett enda anrop kan dela ett värde som kontrollen borde ha läst om.

En värd som ingen har rört rapporterar ändringar vid varje körning. Modulerna `command` och `shell` rapporterar en ändring varje gång de körs, eftersom de inte kan veta vad kommandot gjorde. En playbook full av dem konvergerar aldrig till noll, och verkliga ändringar göms i bruset. Välj hellre en modul som hanterar saken direkt, lägg till `creates` eller `removes`, eller tala om för Ansible när utdata betyder en ändring med `changed_when`.

Ett lösenord dyker upp i en logg eller i repot. Variabler är klartext om inget krypterar dem. Ansible Vault krypterar variabelfiler eller enskilda värden så att de kan ligga i repot, och `no_log: true` på en uppgift håller dess argument och resultat borta från utdata. Kör playbooken en gång med hög detaljnivå i en testmiljö och läs vad den skriver ut innan du litar på någotdera.

Läs mer

- Ansible Community Documentation — projektets egen ingång till installation, begrepp och varje guide.
- Ansible playbooks — plays, uppgifter, moduler, körordning och vad idempotens betyder i praktiken.
- Validating tasks: check mode and diff mode — vad en torrkörning kan och inte kan se, och reglagen för varje uppgift.
- How to build your inventory — värdar, grupper och variablerna som hör till dem.
- Protecting sensitive data with Ansible Vault — att kryptera filer och enskilda värden som ligger bredvid playbooks.
- ansible.builtin.command — referensen för modulen, bl.a. dess delvisa stöd för check mode.

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 Ansible?” 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/ansible/