Oscar Petersson

En LiTHen guide till Ansible

rev. 2018/09/10 01:09:13

Innehåll

Komma igång

1.1 Introduktion

1.1.1 Vad är det här för dokument?

Risken är överhängande att du blivit påtvingad detta dokument av en studiekamrat eller kollega av någon anledning. Troligtvis på grund av att denne insett hur smidigt Ansible är för konfiguration av klienter, alternativt har du hittat det själv och eventuellt undrar du varför du ska fortsätta läsa. Oavsett orsaken så bör vi reda upp några saker:

Den här guiden är inte avsedd att vara en komplett dokumentation för Ansible – det vore redundant, vilket förvisso är bra, men det skulle innebära att jag skulle behöva hålla den uppdaterad frekvent. Inte heller kommer Ansible att hjälpa dig tidsmässigt att konfigurera ett enskilt system. Ju färre system och ju mer sällan de konfigureras, desto mindre nytta har du av Ansible. Men, Ansible är ett utmärkt verktyg för att komma in i "sysadmintänket" (gör inte två gånger det du kan scripta) utan att behöva lära sig shellscript eller Python. Givetvis är det bra att kunna dessa också, men det är tidsödande om man huvudsakligen ska konfigurera vardagliga saker. Den här guiden är främst utvecklad med avseende på kursen TDDI41 vid LiU, och kommer även att använda enskilda delar av kursen som exempel, även om jag kommer att hålla mig ifrån att använda fullständiga exempel. Trots allt så tenderar man att lära sig mest av att sitta och laborera med det själv.

Om det är något du vill göra men som inte täcks här eller i boken oven, googla eller vänd dig till något lämpligt forum på webben. Det den här guiden syftar till är att snabbt få upp nya användare till en nivå där de kan börja använda Ansible för att lösa labbuppgifterna.

Om du springer på något som du känner borde ha tagits upp och som faller inom dokumentets syfte, kontakta mig gärna på guidens gitrepo http://github.com/oscpe262/ansible.guide och berätta.

1.1.1.1 Hur ska jag läsa det här dokumentet?

Det står läsaren givetvis fritt att läsa det precis som denne själv önskar. Ett varningens finger bör dock lyftas till den som ämnar hoppa runt i texten då vissa avsnitt bygger på tidigare avsnitt .

1.1.1.2 Kritik av TDDI41

Jag läste kursen hösten 2016, ett år innan jag började jobba som systemadministratör. Även om kursen det året hade problem framför allt med prestanda på labbsystemet så hade jag inte jättemycket att invända mot innehållet, inte minst eftersom jag inte visste vad jag kunde förvänta mig. Rent innehållsmässigt bjöds det på många bra delar, men även en hel del utdaterade, vilket var bra när man kom ut i verkligheten med en massa förlegade system. Sällan kunde jag ana att det första jag skulle få användning utav var NIS-erfarenhet ...

Nåja, vad som var bristande med kursen var dock vad jag kallar "sysadmintänk". Det första som rekommenderades var att göra alla labbar manuellt först, den sista obligatoriska labben var att scripta allt och

skriva tester, och sedan göra om allt med exempelvis Puppet, Ansible eller dylikt för ett högre betyg. Dumt tänkt!

Skulle jag ha lagt upp det så skulle jag ha gjort tvärt om – använd ett sådant verktyg först, för det är vad du vill använda i verkliga livet om du sysslar med just 'systemadministration'. Ur det perspektivet är Ansible ett utmärkt verktyg, just för att det inte gör jobbet åt dig – du måste förstå vad som ska göras – men det besparar dig en massa scriptande.

1.1.2 Vad är Ansible?

Ansible är en öppen mjukvara för automatisering av bland annat mjukvaruprovisionering och konfigurationsstyrning. Ansible ansluter via SSH (eller PowerShell, eller andra API:er, men jag lägger här fokus vid styrning från och till linuxmaskiner) till en eller flera noder – seriellt eller parallellt.

Det som skiljer Ansible från exempelvis Chef eller Puppet är att Ansible är agentlös. Alla ändringar pushas ut till noderna. Det går att göra hooks, inte minst via Ansible Tower (licensmjukvara), men det är inget som kommer att beröras här.

1.1.3 Vad tjänar jag på att använda Ansible?

Tid och läsbarhet. Tack vare att Ansible skeppas med transparenta moduler så är det lätt att följa YAML-koden över vad varje playbook eller role gör. Likaså är det lätt att skriva dessa och man behöver inte sitta och sköta loopning över noder, loopning över listor på paket, och så vidare. Likaså behöver vi inte fundera på idempotens i större utsträckning om vi gör rätt från början. Det tar givetvis tid att lära sig Ansible, men skalar vi upp saker och ting så kommer det att löna sig.

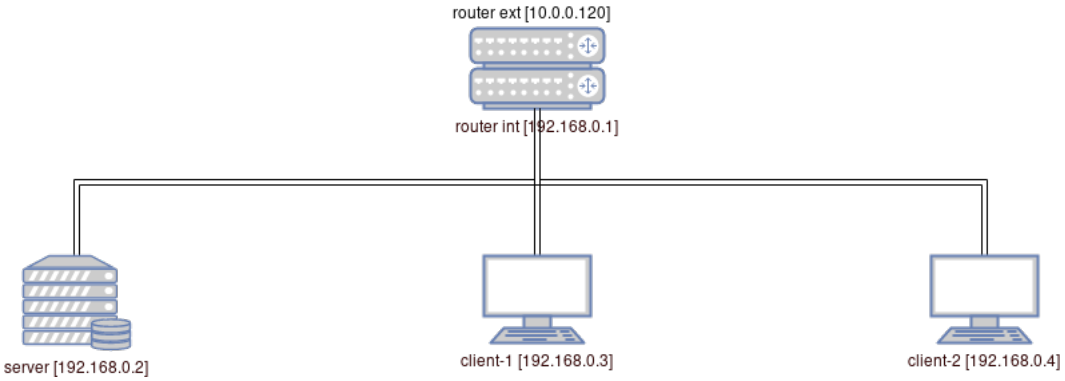
1.1.4 Övrigt

TDDI41 använder sig av debianbaserade OS, vilket är något jag har använt i väldigt låg utsträckning utöver just i den kursen. Jag strävar efter att göra den här guiden så generell som möjligt, och i de fall det inte går kommer jag att vända mig till min dokumentation från TDDI41. Det finns dock en överhängande risk att det kan finnas färgningar av RedHat i det hela, då jag främst jobbar med RHEL och CentOS, samt Fedora privat och på min jobbdator.

Grundläggande körexempel

2.1 Miljön

I det här avsnittet kommer vi att se exempel på hur vi kan använda Ansible för att konfigurera en uppsättning maskiner som ser ut ungefär som den uppsättning vi har i TDDI41. Vi börjar med att titta på strukturen över de fyra maskinerna, inte minst för de som inte är bekanta med kursens innehåll.



Figur 2.1: Överblick av noderna i TDDI41

Bilden är inte helt rättvisande då den refererar till interna ip-adresser (nätmask /24), men det är något vi inte bryr oss nämnvärt vid i det här fallet. Våra exempel kommer även att förenkla det hela än mer och endast se till det inre nätet. Ett alternativt sätt att se på det hela är att vi skulle använda enheten `router` som den maskin vi pushar våra konfigurationer på. Anledningen till denna förenkling är för att fokusera på och tydliggöra själva Ansible-delen snarare än att lösa hela labbserien. Vi kommer även förutsätta att vi har satt IP-adresser på maskinerna innan vi börjar med Ansible.

Enheten `router` är en maskin med dubbla nätverkskort (vi bortser från detta här som sagt) som, precis som namnet antyder, agerar gateway till det inre nätverket samt ntp-server. Enheten `server` är en maskin som agerar fillagring, autentiseringsserver, eventuell mailserver, med mera. Enheterna `client-`*`n`* är i det här avseendet identiska sånär som på namn och ip och är huvudsakligen till för att skala upp miljön.

2.2 Hur vi använder Ansible

När vi vill pusha ändringar till våra noder kan vi använda två olika kommandon: `ansible` samt `ansible-playbook`.

Kommandot `ansible` är det råa kommandot där vi måste specifiera vilka moduler etc. vi skall använda, vilket kan användas för att exempelvis kolla att våra noder är uppe eller för att se variabler som samlas in från en nod. Vi återkommer till detta.

Kommandot `ansible-playbook` är det vi huvudsakligen kommer att använda oss utav när vi faktiskt pushar ändringar. Det kör då en specifik playbook som anger på vilka noder vad ska köras. Som vi kommer se kan vi göra många justeringar på många system samtidigt med hjälp av dessa playbooks.

2.2.1 Vad händer vid en körning

Om vi ser till vad som händer när vi kör en playbook så händer enkelt sett följande: Vi ansluter via SSH till noden, skickar över de moduler tillsammans med variabler för hur dessa ska köras samt när, kör en modul som inhämtar körinformation i form av variabler från noden för att (eventuellt) använda dessa variabler tillsammans med de vi skickat med. Till sist kör vi då det script vi skickat över som använder överskickade moduler för att göra förändringarna på noden, och så skickar vi tillbaka körinformation till styrenheten som skriver ut hur det gick på skärmen.

Komplicerat? Ja, bakom ridån, men inte användningsmässigt. Låt oss titta vidare.

2.3 Ett faktiskt exempel

Vi kommer senare att gå in på hostfilen, syntaxer och så vidare, men låt oss först ta ett exempel där vi visar hur enkelt ett kommando för att konfigurera alla client-noder:

```
$ ansible-playbook clients.yml
```

Skulle vi dessutom vilja ansluta som en särskild användare, begränsa oss till en enda klient, begränsa de tasks vi vill köra till att endast inkludera NTP-konfiguration och dessutom bara testköra (inga förändringar på den faktiska noden), då skulle det kunna se ut som följande:

```
$ ansible-playbook clients.yml -u foouser -k --limit client-1 --tags NTP -C
```

Körkommandot kan i stort sett sköta det mesta, men vanligtvis vill vi ställa in så mycket som möjligt i konfigurationsfilerna, dels Ansibles, men även hostspecifika och gruppspecifika, men mer om det under 4 Best practices.

Filer

I det här avsnittet ska vi titta på de vanligaste konfigurationsfilerna samt viss filstruktur i det upplägg vi senare kommer att beröra i 4 Best practices.

3.1 /etc/ansible

Under `/etc/ansible` finner vi systemkonfiguration för styrsystemet samt globalt tillgängliga inventories, roles etc. Vi kommer dock inte att använda mer än `ansible.cfg` och `hosts` på en global nivå.

3.1.1 ansible.cfg

Generellt behöver inte den här filen modifieras, men det finns vissa saker som kan vara av intresse även på en nybörjarnivå.

`remote_user` ger oss möjligheten att ansluta med en specifik användare. Vi kommer att beröra detta mer under best practices.

`log_path` kan vara bra att sätta till en lämplig sökväg (ex. `/var/log/ansible.log`) för att aktivera loggning..

`vault_password_file` ger möjligheten att ange en sökväg för en så kallad 'vault file' - en krypterad fil där man kan spara variabler man inte vill ha liggande för allmän åskådan. En sådan fil kan även åkallas på commandline med `--vault-password-file`.

`ansible_managed` är en variabel med samma namn. Denna brukar användas i konfigurationsfiler för att markera att dessa är konfigurationsstyrda och således inte bör ändras direkt på systemet då dessa ändringar förr eller senare kommer att skrivas över.

`nocows` styr kor. Rekommenderas att sätta till '1' om man inte hyser bovina läggningar.

3.1.2 hosts

`/etc/ansible/hosts` är den globala host-listan (inventory). Man kan (och eventuellt bör, beroende på miljön i helhet) använda separata hosts-filer och åkalla dessa på kommandoraden.

Inventories är vanligen skrivna i ett init-liknande format, men kan i senare versioner av Ansible även vara i YAML-format. Vi kommer här att använda det klassiska. Vi kommer att ta vårat exempel från TDDI41 för att visa hur en sådan kan se ut.

Det vi ser är en enkel enumrering av noderna där vi ger dem funktionella grupperingar (första blocket) där vi kopplar ihop IP-adresser till dessa grupperingar. Vi skulle lika gärna kunna använda domännamn bör dock sägas, och skulle vi ha ännu fler noder skulle vi kunna fortsätta lista dessa under grupperingarna. För att demonstrera detta listar vi även ett subset inaktiva (utkommenterade) clients som inte är aktuella än.

Värt att notera är även att vi kan använda ranges (:). I det här fallet har vi gett ett spann IP-adresser, men skulle vi ha använt oss utav domännamn skulle vi även ha kunnat använda oss av ranges i dessa, exempelvis `client-[1:2]`

```
### BOF ###
## /etc/ansible/hosts
#
## Enumerating nodes

[gateway]
192.168.0.1

[server]
192.168.0.2

[clients]
192.168.0.[3:4]
#192.168.1.[1:50]

## Groupings

[internal:children]
gateway
server
clients

[authnodes:children]
server
clients
### EOF ###
```

Best practices

"Best practice" är ett begrepp som alla har en bild utav vad det innebär inom ens eget område. Tyvärr är det i stort sett aldrig är så att det finns en enhetlig bild inom varje område, sällan ens på varje arbetsplats, eller ens mellan team på samma arbetsplats.

När det gäller systemadministration så är det givetvis likadant, och det viktigaste är i sedvanlig ordning dokumentation. Har man en god dokumentation så kan resten lösas med resurser. Ansible är inte avsett att vara ett dokumentationsverktyg, men det kan utgöra ett utmärkt komplement till exempelvis en wiki.

Men, det här är inte en guide till systemadministration eller dokumentation, det är en guide till Ansible, så låt oss fokusera på detta.

Best practice är till viss del beroende på hur vad man använder Ansible till och hur stora och varierade miljöer man har. Den här guiden försöker främst hålla ett scenario tanken där vi har en varierad miljö med olika linuxdistributioner och där vi samtidigt söker en skalbarhet. Att hantera en eller ett par maskiner klarar man utan att blanda in exempelvis Ansible (även om det är praktiskt att kunna rulla om maskiner smidigt), men att hålla tiotals, hundratals, eller till och med tusentals klienter i ordning på ett enhetligt sätt kräver verktyg, och varje miljö börjar med en första maskin.

Att använda någon form av versionshantering är att se som lite av ett måste när samarbete mellan administratörer förekommer, och även en automatiseringskedja när miljön växer. Det är dock två faktorer vi inte kommer att beröra här då det är utanför denna guides "scope", men möjligheterna finns.

4.1 Filstruktur

En tydlig filstruktur hjälper skalbarhet, läsbarhet och även implementation av versionshantering (ex. Git). Det finns flera olika modeller som är bra, men här lyfts bara en av den enkla anledningen att det är en modell undertecknad själv har använt framgångsrikt och för att omfattningen av denna guide skall hållas begränsad. Den är en av de modeller som rekommenderas av RedHat, så det är inget hemkok på något sätt. Vi börjar med att illustrera den:

```
./
  hosts                   # global inventory file

  group_vars/
    <groupname>           # variables for groups
  host_vars/
    <hostname>            # variables for specific systems

  library/                # custom modules
  filter_plugins/         # custom filter plugins

  playbooks/              # directory for playbooks
    playbook1.yml
```

```
roles/
  common/               # this hierarchy represents a "role"
      tasks/            #
          main.yml      #  <-- tasks file can include smaller files if warranted
      handlers/         #
          main.yml      #  <-- handlers file
      templates/        #  <-- files for use with the template resource
          ntp.conf.j2   #  <------- templates end in .j2
      files/            #
          bar.txt       #  <-- files for use with the copy resource
          foo.sh        #  <-- script files for use with the script resource
      vars/             #
          main.yml      #  <-- variables associated with this role
      defaults/         #
          main.yml      #  <-- default lower priority variables for this role
      meta/             #
          main.yml      #  <-- role dependencies
```

Inventory-filer har vi redan berört, och generellt bör endast en sådan finnas och hållas så läsbar som möjligt. Det är även i inventoryt som grupperingar av maskinfunktioner bör ske på ett vettigt sätt.

`library` samt `filter_plugins` är för egenskrivna moduler och plugins. Det är inget vi kommer att beröra här dock.

`group_vars` och `host_vars` innehåller variabelfiler för nodgrupper och enskilda noder. Här bör vi tänka på att vi vill lägga så mycket specifik konfiguration av enskilda maskiner/grupper här.

`roles` är vad vi skulle kunna likna vid programmeringsfunktioner och strukturer. Roller bör vara så generellt skrivna som möjligt, utan nod-specifika variabler, idempotenta, begränsade i dess funktion och om möjligt oberoende av distribution (eller med stöd för alla berörda distributioner).

`playbooks` är den mapp där vi generellt lagrar de playbooks vi skriver. En playbook kan vara hur begränsad eller omfattande man önskar, men generellt bör de vara begränsade antingen i funktion eller gruppering av noder. Kan man inte på mindre än en minut få en god överblick över vad en playbook gör bör man fundera på hur man kan bryta ner den i mindre.