GIT - Infrastructure as code, Student Workbook

Mathias Homann

11-2021

GIT - Infrastructure as code

Student manual

Was bedeutet eigentlich "Infrastructure as code"?

Immer öfter hört man heute den Begriff "Infrastructure as code", aber was bedeutet das eigentlich?

Es ist eigentlich ziemlich simpel: "Infrastructure as code" ist nichts anderes als eine neue Art und Weise, an die Konfiguration kompletter IT-Umgebungen heranzugehen. Dabei wird so viel wie möglich in Form von Scripten (im weitesten Sinne des Wortes) implementiert. Dieses bringt uns mehrere Vorteile:

- Wiederholbarkeit
- Fehlerfreiheit
- Automatisierbarkeit

Wiederholbarkeit bedeutet, dass man den gleichen Prozess immer wieder verwenden kann, und sich darauf verlassen kann dass der Prozess immer gleich abläuft.

Fehlerfreiheit bedeutet, dass ein Prozess, den man ein mal korrekt implementiert hat, durch seine Wiederholbarkeit sicherstellt, dass keine Fehler auftreten.

Automatisierbarkeit ist die Kombination dieser Eigenschaften: Einen Prozess, den ich korrekt und wiederholbar implementiert habe, kann ich mit gutem Gewissen automatisch ablaufen lassen.

Idempotenz

Eine wichtige Voraussetzung hier ist die sogenannte **Idempotenz**.

Idempotenz bedeutet, dass ich einen Prozess beliebig oft wiederholen kann, und garantiert jedesmal das gleiche Ergebnis herauskommt. Ein Beispiel aus der Welt der Zahlen:

`x=x+2` ist **nicht** idempotent - nach dem ersten Aufruf ist der Wert der Variablen 2, danach 4, dann 6, und so weiter.

`x=x*2` ist hingegen idempotent - egal wie oft man es ausführt, das Ergebnis ist **immer gleich**.

Um das ganze mal in die Welt der Systemadministration zu heben:

`mkdir /tmp/testdir` wird beim ersten Aufruf funktionieren, aber wenn das Verzeichnis dann existiert, wird jeder weitere Aufruf zu einer Fehlermeldung führen - also **nicht** idempotent.

`test -d /tmp/testdir || mkdir /tmp/testdir` wird hingegen bei **jedem** Aufruf ordnungsgemäß funktionieren - also ist diese Form **idempotent**.

Systemunabhängigkeit

Eine weitere wichtige Eigenschaft der verwendeten Skripte ist die **Systemunabhängigkeit**.

Das bedeutet, dass man in den Skripten beschreibt, welches Ergebnis man wünscht, aber sich nicht darum kümmern muss, wie genau dieses Ergebnis zu erreichen ist - darum kümmern sich die verwendeten Werkzeuge.

Versionsverwaltung

Zu guter Letzt benötigt man eine wie auch immer geartete **Versionsverwaltung** in der die erzeugten Konfigurationsscripte abgelegt werden. Dadurch kann man immer zur letzten als gut bekannten Konfiguration zurückkehren wenn es denn doch einmal nötig sein sollte.

Welche Tools gibt es

Automatisierungstools

- Saltstack: Ein open source Konfigurationsmanagement für Server. Setzt einen zentralen Server voraus, der die Aufgaben an die "minions" verteilt. Steht unter der apache-Lizenz. Wurde in 2020 von VMWare aufgekauft, seitdem ist die "offizielle Webseite" ziemlich nutzlos.
- Puppet: Ein weiteres open source Konfigurationssystem. Auch puppet benötigt einen zentralen Server (den "Puppet master"). Wichtiges Merkmal: Die verschiedenen schritte in puppet-"scripten", den sogenannten Modulen, können, und werden, in beliebiger Reihenfolge ausgeführt werden. **Idempotenz** ist hier **lebenswichtig**.
- Ansible: Das dritte der verbreiteten Konfigurationssysteme. Ansible-Playbooks sind in YAML verfasst, einfach zu lesen, und werden (im Gegensatz zu Puppet) von oben nach unten abgearbeitet, was den Umgang mit

Ansible wesentlich einfacher macht. Ansible benötigt im Gegensatz zu Puppet und Saltstack **keinen** zentralen Server, es gibt aber eine zentralisiert einsetzbare Webapplikation, den "Ansible Tower", der Ansible um RBAC (role based access controls), Logging, scheduling, u.v.m. erweitert.

Red Hat Linux unterstützt Puppet und Ansible, wir werden uns im Weiteren mit Ansible befassen.

Versionsverwaltung

Versionsverwaltungssoftware gibt es wie Sand am Meer: rcs, cvs, mercurial, subversion, git, um nur die bekanntesten zu nennen. Wir werden im weiteren auf git näher eingehen.

Was ist Ansible

Ansible ist eine "Systembeschreibungssprache", das bedeutet ich sage was ich will, und Ansible weiss dann "von selber" wie das zu machen ist. ein Beispiel:

`ansible -m package -a 'name=httpd state=latest' -bkK server17` heißt in umgangssprachlichem Deutsch: "Liebes Ansible, ich möchte, dass auf Server17 das Paket 'httpd' in der aktuellsten Version installiert ist. Dazu brauchst du das `package` modul, das musst du als root machen, nach den benötigten Passwörtern sollst du mich fragen." Und ob Server17 nun ein RHEL (oder ein Abkömmling) ist, wo man das entweder mit yum oder dnf macht, oder ein SLES/openSUSE wo der Paketmanager zypper heisst, oder gar ein debian mit apt-get oder aptitude - das ist vollkommen Wurst, darum kümmert sich Ansible selber. Und auch ob das Paket schon drauf ist, oder in einer älteren Version, oder gar nicht, ist für diesen Befehl egal.

Mit ansible kann man (wie eben gesehen) sogenannte "ad-hoc-befehle" ausführen lassen, oder man schreibt sogenannte "playbooks". Ein Playbook ist eine ganze Abfolge von ansible-modulaufrufen, die nacheinander ausgeführt werden.

Und was ist mit der Idempotenz?

Ansible-befehle sind bis auf wenige Ausnahmen immer idempotent - wenn in dem obigen Beispiel das httpd-paket schon in der aktuellsten Version installiert ist sagt Ansible einfach "Oh. Na dann ist ja alles gut und ich muss nix tun." Die wenigen Ausnahmen: alles was man mit dem -raw, dem -command und dem -shell modul macht. Mit diesen drei Ansible-Modulen werden nämlich direkt "native" Befehle auf den Zielsystemen ausgeführt - ohne auf die Ergebnisse zu achten. Somit muss man da dann selber auf die Idempotenz achten... und das kann schon etwas fehleranfällig werden.

Ansible-module?

Ansible basiert auf tausenden von Modulen. In diesen Modulen steckt die eigentliche Logik hinter Ansible. So z.B. das oben schon verwendete "package" Modul, was betriebssystemunabhängig Softwarepakete installiert, deinstalliert, und updatet. Es gibt zur Zeit über 3300 Module für alle Zwecke, vom einfachen Anlegen von Dateien bis zur Verwaltung meiner AWS-Instanzen, und viele mehr - sogar Windows-Systeme und Netzwerkhardware kann mit Ansible gemanagt werden.

Eine liste aller für meine Ansible-version verfügbaren Module sehe ich mit dem Befehl `ansible-doc -l`. Mit dem gleichen Befehl sehe ich dann auch die Dokumentation für ein Modul, z.B. so: `ansible-doc package` um die Doku für das package-modul zu lesen.

Playbooks?

Ein Ansible-playbook ist im Grunde nichts anderes als ein Script, in dem nacheinander verschiedene Module mit Parametern aufgerufen werden. Playbooks sind in YAML geschrieben ("yet another markup language"). Zu YAML muss ich zwei Dinge wissen:

- YAML ist einfach wie Klartext zu lesen (das ist gut)
- YAML ist sehr pingelig was die Einrücktiefe angeht - eine falsch eingerückte Zeile führt zu Fehlermeldungen - aber nicht unbedingt in der betreffenden Zeile sondern ganz wo anders... (Das ist nicht gut)

Für die Problematik mit den Einrückungen gibt es eine relativ einfache Lösung:

- den `vim` Editor benutzen
- die folgende Zeile in `~/.vimrc` eintragen:

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et cc=3,5,7,9,11
```

Diese Zeile führt dazu, dass `vim` bei YAML-Dateien automatisch einrückt, das immer um zwei Leerzeichen mehr (oder weniger) tut, und dazu noch die ersten 5 Einrückstufen in rot markiert. Dieser Trick hat dem Ersteller dieses Dokumentes auch schon viele graue Haare erspart...

Zuletzt noch ein Beispiel für ein sehr einfaches Playbook:

```
- hosts: test-servers
  remote_user: nahmed
  become: true
  vars:
    project_root: /var/www/html
  tasks:
  - name: Install Apache Webserver
    yum: pkg=httpd state=latest
  - name: Place the index file at project root
```

```
    copy: src=index.html dest={{ project_root }}/index.html owner=apache group=apache mode=(
  - name: Enable Apache on system reboot
    service: name=httpd enabled=yes
    notify: restart apache
  handlers:
  - name: restart apache
    service: name=httpd state=restarted
```

Was genau passiert hier nun? Zuerst werden einige Daten definiert: die "Zielhosts" sind alle Hosts die im Inventory in der Gruppe "test-servers" sind. Der Benutzer für den Zugang ist "nahmed" und alle aktionen sollen dann als root ausgeführt werden (become: true). Danach wird eine Variable definiert, und dann folgen die Tasks und handlers (Ein handler ist ein task, der nur dann ausgeführt wird, wenn er von irgend einem anderen Task ein "notify" erhalten hat).

Der erste Task stellt sicher, dass das httpd paket in der neuesten Verion installiert ist. Der zweite Task kopiert eine Datei, die im gleichen Verzeichnis wie das playbook liegt, an eine bestimmte Stelle auf allen betroffenen Servern. Der dritte Task stellt sicher, dass der httpd-dienst beim boot aktiv ist, und schickt ein notify an den einzigen handler in diesem playbook. Der handler stellt dann zuletzt sicher, dass der httpd-dienst neu gestartet wird.

. . . das inventory?

Ansible benötigt ein "inventory". Das ist im allereinfachsten Fall eine Textdatei, in der Zeile für Zeile die Hostnamen aller Systeme aufgelistet sind, mit denen man arbeiten will, also z.B. so:

```
server1.local.net
server2.local.net
dbserver1.local.net
dbserver2.local.net
...
```

Und wo liegt das Inventory? oder: die Datei ansible.cfg

Daneben gibt es noch die Datei ansible.cfg, in der (unter Anderem) festgelegt wird, wo genau das Inventory zu finden ist. Jedesmal wenn ich ansible aufrufe, wird an mehreren Stellen nach so einer Datei `ansible.cfg` gesucht. 1. in `/etc/ansible/ansible.cfg` 2. in `~/.ansible.cfg` 3. in `./ansible.cfg`

Das heisst also: in /etc kann ich (als Administrator) globale Vorgaben für Ansible machen. Als User mache ich die Vorgaben für mich alleine in `~/.ansible.cfg` festlegen. und dann gibt es zu jedem meiner Ansible-projekte die Möglichkeit eine lokale Datei `ansible.cfg` anzulegen.

Ein Beispiel für eine ~/.ansible.cfg:

```
[defaults]
```

```
inventory = /home/mathias/.ansible/inventory/
remote_user = ansible-remote
private_key_file = /home/mathias/.ansible/id_rsa_ansible
```

Was man hier sieht, bedeutet: * es gibt ein default inventory im ordner /home/mathias/.ansible/inventory/ (es werden einfach alle Dateien in dem Ordner der Reihe nach als Inventory-Dateien eingelesen) * Der Benutzer, als der Ansible sich per ssh an den Zielsystemen anmeldet, heisst `ansible-remote` * Der ssh-key für den Zugang liegt in `/home/mathias/.ansible/id_rsa_ansible`

Vorbereitungen für Ansible:

Aus dem bis hier gesagten ergeben sich folgende Arbeiten, um einen (neuen) REchner für Remote-Administration mit Ansible einzurichten:

1. den Remote-User anlegen
2. den Remote-User für `sudo` freischalten
3. den für Ansible verwendeten ssh-key im Benutzeraccount des ansible-remote-users eintragen.

Und das kann man dann natürlich mit ansible machen:

```
---
- name: create the remote user for ansible and awx
  hosts: all

  tasks:

  - name: create ansible user
    user:
      name:         ansible-remote
      system:       yes
      state:        present
      password:     "{{ upassword | password_hash('sha512') }}"
      create_home:  yes
      expires:      -1

  - name: install ssh authorized key for ansible-remote account
    authorized_key:
      comment: ansible-remote key for ansible and awx
      exclusive: true
      key: "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDOyicC4KQN3yxW7qI5fC1wLra05vCJNjcPhv5b9unF
      state: present
      user: ansible-remote

  - name: comment out "targetpw" in sudoers
    replace:
      path: /etc/sudoers
```

```
    regexp: '(^Defaults targetpw)'
    replace: '## \1'
    validate: /usr/sbin/visudo -cf %s

- name: comment out blanket ALL ALL in sudoers
  replace:
    path: /etc/sudoers
    regexp: '(^ALL.*$)'
    replace: '## \1'
    validate: /usr/sbin/visudo -cf %s

- name: install sudo rule for ansible-remote user
  copy:
    dest: /etc/sudoers.d/ansible-remote
    content: "ansible-remote ALL=(ALL) ALL"
    mode: 0640
    group: root
    owner: root
    validate: /usr/sbin/visudo -cf %s
```

Aufgerufen wird dieses playbook dann so: `ansible-playbook -u root -l zielhost -e upassword=dideldum ansibleuser.yml`. Damit wird der remote-benutzer angelegt, und als passwort das mit "upassword=" übergebene Passwort eingetragen.

Mit den bis hier erfolgten Ausführungen sollte man in der Lage sein, dieses Play zu lesen und zumindest in Grundzügen verstehen.

Die für die Übungen bereitgestellten VMs sind bereits entsprechend vorbereitet.

Zugangsdaten der VMs

Der user für den Benutzerzugriff heißt **`student`** mit dem passwort **`student`**

Das Passwort für den Benutzer **`root`** lautet **`Funk3nGr00v3n123`**

Versionsverwaltung mit git

Eine wie auch immer geartete Versionsverwaltung sollte die folgenden Merkmale aufweisen:

- Jede Änderung wird mit einer Logmessage gespeichert
- Dateien können wieder auf frühere Versionen zurückgesetzt werden
- Die Unterschiede zwischen verschiedenen Versionen können übersichtlich dargestellt werden
- Bei jeder Änderung wird Uhrzeit, Datum, und der Account der die Änderung gemacht hat gespeichert.

- Viele verschiedene Nutzer können gleichzeitig am gleichen Projekt arbeiten, wobei Konflikte die entstehen wenn mehrere Nutzer die gleiche Datei ändern abgefangen bzw. sauber zusammengeführt werden.
- Es gibt die Möglichkeit das Versionskontrollsystem über sog. `hooks` in den Workflow der Softwareentwicklung zu integrieren, so kann z.B. das Erstellen von neuen Containerimages automatisiert werden (gut zu sehen auf dockerhub für zahlende Accounts, oder auf quay.io)

Was ist git

`git` ist eines der weitestverbreiteten Versionskontrollsysteme dieser Tage. Das kommt wohl nicht zuletzt daher, dass einerseits der Linux-Kernel in einem GIT-repository gepflegt wird, und andererseits unter `https://github.com` ein großer öffentlicher GIT-server für jeden der es nutzen will zur Verfügung gestellt wird.

Git arbeitet mit bis zu vier verschiedenen Arbeitsbereichen:

- central repository
- local repository
- staging
- working tree

Das "central repository" liegt üblicherweise auf einem zentralen Server. Das kann der bereits erwähnte Dienst github.com sein, man kann aber auch sehr einfach im lokalen Netz einen eigenen zentralen git server aufsetzen, wie wir noch sehen werden. Dieses "central repository" bezeichnet man gerne als den **upstream** eines Projektes.

Das "local repository" ist die lokal ausgecheckte Kopie eines upstreams.

Innerhalb des "local repositories" gibt es zwei Bereiche: die "staging area" in die alle Änderungen eingetragen sind, die **noch nicht committed** sind, und den "working tree", d.H. die regulären Dateien im Dateisystem.

Wie benutze ich git

Um mit git zu arbeiten, folge ich einem einfachen Zyklus: **change, stage, commit, push, repeat**

Zuerst wird ein repository angelegt. Das geschieht entweder lokal oder auf einem zentralen server mit `git init`. Dabei werden auf einem zentralen server noch die parameter `--bare --shared=true` angegeben.

Alternativ dazu kann man auch ein bereits existierendes Repository von einem zentralen server kopieren ("clonen"): `git clone REPOURL`

Nun macht man Änderungen. Wenn man einen Zwischenstand im Repository ablegen will, fügt man die geänderten Dateien mit `git add DATEINAME` zum staging-Bereich hinzu (Dies gilt auch für neuangelegte Dateien).

Um den im staging abgelegten Stand ins Repository zu schreiben, verwendet man `git commit`, wobei die commit-message mit `-m "message"` gleich übergeben werden kann.

Zuletzt kann man alle im lokalen Repository abgelegten Änderungen mit `git push` ins zentrale Repository hochladen.

Vorbereitende Aufgaben

Die Lab-Umgebung

Für jeden Teilnehmer des Workshops stehen zwei separate VMs zur Verfügung, auf denen es einen Benutzer "student" mit dem Passwort "student" gibt. Die VMs haben für alle Teilnehmer die gleichen hostnamen, `server` und `workstation`. Auf der Workstation ist eine graphische Benutzeroberfläche eingerichtet, die ggf. erst an die Sprach- und Tastaturpräferenzen des Benutzers angepasst werden muss. Das Kontrollzentrum des verwendeten DE ist selbsterklärend.

ssh key anlegen

Öffne eine Kommandozeile auf der Workstation-VM, und erzeuge einen ssh-key mit dem Befehl `ssh-add`:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/student/.ssh/id_rsa):
Created directory '/home/student/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/student/.ssh/id_rsa.
Your public key has been saved in /home/student/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:B4ytYCABkw3JNKVTluqcXSSs4M1E9a6DJaDeqJidkZc Der SSH KEy für den Ansible-Workshop
The key's randomart image is:
+---[RSA 3072]----+
|XO==o.           |
|++** ..+         |
|o+* = ..+        |
|.+o+ o.. .       |
|+ o.....S .      |
|.+oo+..  .       |
| oooEo           |
|oo +  .          |
|+ o              |
+----[SHA256]-----+
```

Im hier gezeigten Beispiel hat dieser key **keine** passphrase. **Dies ist natürlich im 'Wahren Leben' keine gute Idee, macht es aber hier einfacher.

ssh key auf server und workstation kopieren

Diesen Key müssen wir nun auf server und workstation für die benutzer `root` und `student` freischalten. Dies geschieht mide dem Befehl `ssh-copy-id`. Zuerst für den student:

```
[student@workstation ~]$ ssh-copy-id student@server
The authenticity of host 'server (192.168.238.174)' can't be established.
ECDSA key fingerprint is SHA256:PFZ8jvuycTzUHv1Eb8DhvWyDRo4qBz4lH48wLJY33XQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
/usr/bin/ssh-copy-id: INFO: 2 key(s) remain to be installed -- if you are prompted now it is
student@server's password:

Number of key(s) added: 2

Now try logging into the machine, with:   "ssh 'student@server'"
and check to make sure that only the key(s) you wanted were added.
```

Diesen Vorgang wiederholen wir noch drei mal: `ssh-copy-id root@server`, `ssh-copy-id student@workstation` und `ssh-copy-id root@workstation`.

Zum Testen können wir uns dann mit `ssh root@server` auf den Server anmelden, analog dazu auch als student auf server oder workstation.

git repo auf server anlegen

Jetzt können wir auf dem server als der student-user ein gemeinsam genutztes git repository anlegen:

```
[student@server ~]$ mkdir ansible-mysql
[student@server ~]$ cd ansible-mysql/
[student@server ansible-mysql]$ git init --bare --shared=true .
Leeres verteiltes Git-Repository in /home/student/ansible-mysql/ initialisiert
[student@server ansible-mysql]$ ls -la
insgesamt 16
drwxrwsr-x. 7 student student  119  2. Dez 08:43 .
drwx------. 4 student student   95  2. Dez 08:42 ..
drwxrwsr-x. 2 student student    6  2. Dez 08:43 branches
-rw-rw-r--. 1 student student  126  2. Dez 08:43 config
-rw-rw-r--. 1 student student   73  2. Dez 08:43 description
-rw-rw-r--. 1 student student   23  2. Dez 08:43 HEAD
drwxrwsr-x. 2 student student 4096  2. Dez 08:43 hooks
drwxrwsr-x. 2 student student   21  2. Dez 08:43 info
drwxrwsr-x. 4 student student   30  2. Dez 08:43 objects
drwxrwsr-x. 4 student student   31  2. Dez 08:43 refs
```

git repo clonen

Dieses repository clonen wir auf die Workstation:

```
[student@workstation ~]$ git clone git+ssh://student@server/home/student/ansible-mysql
Klone nach 'ansible-mysql' ...
warning: Sie scheinen ein leeres Repository geklont zu haben.
[student@workstation ~]$ cd ansible-mysql/
[student@workstation ansible-mysql]$ ls -la
insgesamt 4
drwxr-xr-x.  3 student student   18 29. Nov 03:57 .
drwx------. 16 student student 4096 29. Nov 03:57 ..
drwxr-xr-x.  7 student student  119 29. Nov 03:57 .git
```

ansible.cfg und inventory anlegen

Jetzt können wir eine Konfigurationsdatei und ein Inventory für ansible anlegen:

```
[student@workstation ~]$ cd ~/ansible-mysql/
[student@workstation ansible-mysql]$ vim ansible.cfg
[student@workstation ansible-mysql]$ cat ansible.cfg
[defaults]
inventory = ./inventory/
remote-user = student

[privilege_escalation]
become = False
become_method = sudo
become_user = root
become_ask_pass = False
```

```
[student@workstation ansible-mysql]$ mkdir ./inventory/
[student@workstation ansible-mysql]$ vim ./inventory/000-hosts
[student@workstation ansible-mysql]$ cat ./inventory/000-hosts
server
workstation
```

ein erster Test

Um zu testen ob das alles in Ordnung ist, können wir ein ansible “ad hoc”-Kommando verwenden.

```
[student@workstation ansible-mysql]$ ansible --list-hosts all
  hosts (2):
    server
    workstation
[student@workstation ansible-mysql]$ ansible -m ping server
server | SUCCESS => {
    "ansible_facts": {
```

```
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

ansible.cfg und inventory ins repo pushen

Unsere Konfigurationsdatei und das Inventory scheinen ja in Ordnung zu sein, also ab in's Repository damit, aber dafür muss git erst mal für die Logeinträge wissen wer wir sind:

```
[student@workstation ansible-mysql]$ git config --global user.email Mathias.Homann@techdata.
[student@workstation ansible-mysql]$ git config --global user.name "Mathias Homann"
```

Hier setzt man natürlich den eigenen Namen ein, und die passende Email-Adresse. Wenn man das `--global` weglässt, gilt diese Information nur für die lokale repo-kopie in der man gerade arbeitet.

Jetzt können wir unsere Änderungen speichern:

```
[student@workstation ansible-mysql]$ git status
Auf Branch master

Noch keine Commits

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit vorzumerken)
        ansible.cfg
        inventory/

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
[student@workstation ansible-mysql]$ git add ansible.cfg
[student@workstation ansible-mysql]$ git commit -m "ansible configuration file"
[master (Root-Commit) 0688255] ansible configuration file
 1 file changed, 10 insertions(+)
 create mode 100644 ansible.cfg
[student@workstation ansible-mysql]$ git add inventory/
[student@workstation ansible-mysql]$ git commit -m "ansible inventory"
[master 29c7e81] ansible inventory
 1 file changed, 3 insertions(+)
 create mode 100644 inventory/000-hosts
[student@workstation ansible-mysql]$ git push
Objekte aufzählen: 7, fertig.
Zähle Objekte: 100% (7/7), fertig.
Delta-Kompression verwendet bis zu 2 Threads.
Komprimiere Objekte: 100% (4/4), fertig.
```

```
Schreibe Objekte: 100% (7/7), 658 Bytes | 658.00 KiB/s, fertig.
Gesamt 7 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
To git+ssh://server/home/student/ansible-mysql
 * [new branch]      master -> master
[student@workstation ansible-mysql]$ git log
commit 29c7e81c23d582ee7ad8fe0a5665dfe793681907 (HEAD -> master, origin/master)
Author: Mathias Homann <mhomann@redhat.com>
Date:   Mon Nov 29 04:15:26 2021 -0500

    ansible inventory

commit 06882552e8bfd56e4e3228dec0bc7131c7dbd6c5
Author: Mathias Homann <mhomann@redhat.com>
Date:   Mon Nov 29 04:15:08 2021 -0500

    ansible configuration file
[student@workstation ansible-mysql]$
```

Datenbankadministration mit ansible am Beispiel von mysql auf Red Hat Linux

Was muss alles erledigt werden:

- mysql-pakete müssen installiert werden
- mysql service muss aktiviert werden
- mysql port muss auf der firewall durchgelassen werden, wenn gewünscht
- ein datenbank-user muss angelegt werden
-

  Welche Ansible-Module braucht man

  package mysql_* service firewalld

Welche Schritte müssen ausgeführt werden

Hinweise: * `ansible-doc -l | grep mysql` zeigt alle ansible module, die mit mysql umgehen, danach liest man die entsprechende Dokumentation einfach mit `ansible-doc mysql_user` * das `package`-Modul kann mit so ziemlich allen Paketmanagern unter Linux umgehen, es gibt aber auch spezialisierte Module für die einzelnen Paketmanager der verschiedenen Linuxversionen. * Wir haben auf den VMs Centos 8.2, dort heißt das Paket für den mysql server `mariadb-server` und das Paket mit dem client (zum testen) `mariadb`, und der Paketmanager heisst `dnf`. * Als Firewall kommt `firewalld` zum Einsatz, also brauchen wir dazu das Ansible-Modul `firewalld`.

Schritt für Schritt-Lösung

Zuerst einmal müssen wir die nötigen Pakete installieren, den dienst starten, und auf der Firewall den Port öffnen.

Das folgende Playbook erledigt das für uns.

```
---
- name: update DNF cache on all hosts
  hosts: all
  user: student
  become: true
  tasks:

  - name: Update DNF Package repository cache
    dnf:
      update_cache: True

- name: install and enable mysqld on server
  hosts: server
  user: student
  become: True
  tasks:

  - name: Install MySQL server on CentOS 8
    dnf:
      name: mariadb-server
      state: present

  - name: Install MySQL client on CentOS 8
    dnf:
      name: mariadb
      state: present

  - name: Make sure mysqld service is running
    service:
      name: mariadb
      state: started
      enabled: True

  - name: Install python3-PyMySQL library
    dnf:
      name: python3-PyMySQL
      state: present

- name: install mysql client on workstation
  hosts: workstation
```

```
  user: student
  become: true
  tasks:

  - name: install mysql client
    dnf:
      name: mariadb
      state: present

- name: open mysql firewall port on server
  hosts: server
  user: student
  become: true
  tasks:

  - name: open firewall port for mysql
    firewalld:
      service: mysql
      state: enabled
      permanent: yes
      immediate: yes
```

Aufgerufen wird es mit `ansible-playbook -kK <dateiname>`. `-kK` teilt ansible dabei mit, uns nach dem sudo passwort und der ssh-passphare zu fragen. Bei Verwendung des ssh-agent (oder wie hier im Beispiel oben, eines ssh Schlüssels ohne Passphrase) kann das kleine `k` weggelassen werden.

Unter den meisten Linuxvarianten lässt mariadb / mysql zugriffe als admin direkt nach der Installation erst mal ohne passwort zu. Dagegen hilft das zweite Playbook:

```
---
- name: configure the database server
  hosts: server
  user: student
  become: true
  vars_files:
    - vars/main.yml
  tasks:

  - name: set password for root access
    mysql_user:
      login_host: "localhost"
      login_user: "root"
      login_password: ""
      user: "root"
      password: "{{ mysql_root_password }}"
```

```
      state: present
```

Hier gibt es zwei Besonderheiten zu beachten: * Hier wird die **Idempotenz** verletzt - dieses Play funktioniert nur, wenn noch kein Passwort für `root` in der Datenbank gesetzt ist. * Es wird eine Variablendatei eingelesen. Diese Datei liegt im Unterordner `vars` und hat den Inhalt: `mysql_root_password: Sup3rS3cur3  mysql_dbuser_password: N0tS0S3cur3`

Nun können wir endlich den Datenbankuser anlegen, Rechte vergeben, und eine leere Datenbank anlegen:

```
---
- name: configure the database server, part two
  hosts: server
  user: student
  become: true
  vars_files:
    - vars/main.yml
  tasks:

  - name: create db user
    mysql_user:
      login_host: "localhost"
      login_user: "root"
      login_password: "{{ mysql_root_password }}"
      user: "dbuser"
      host: "%"
      priv: "dbuser_%.*:ALL"
      password: "{{ mysql_dbuser_password }}"
      state: present

  - name: create a demo database
    mysql_db:
      login_host: "localhost"
      login_user: "root"
      login_password: "{{ mysql_root_password }}"
      name: "dbuser_test1"
      state: present
```

Wir verwenden hier wieder die gleiche Variablendatei!

Eleganter ist das mit Rollen

Es geht allerdings viel einfacher: mit einer ansible role, oder Rolle.

Was ist eine Rolle

Ansible-roles sind vorgefertigte "pakete" mit plays die man verwenden kann um die eigenen playbooks viel kürzer und übersichtlicher zu gestalten.

Ansible roles findet man auf/mit ansible galaxy, https://galaxy.ansible.com/

Dort (oder mit google) findet sich eine Ansible-Role zum Installieren und konfigurieren eines mysql oder mariadb datenbankservers von Jeff Geerling (Verfasser von "Ansible for DevOps", sehr aktiv auf youtube): https://galaxy.ansible.com/geerlingguy/mysql

wie installiere ich Rollen

Um eine solche role zu verwenden muss ich die role erst einmal herunterladen und installieren. Dies geschieht mit dem Befehl `ansible-galaxy`:

```
[root@workstation ~]# ansible-galaxy install geerlingguy.mysql
- downloading role 'mysql', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-mysql/archive/3.3.2.tar
- extracting geerlingguy.mysql to /root/.ansible/roles/geerlingguy.mysql
- geerlingguy.mysql (3.3.2) was installed successfully
```

So eine Rolle muss auf dem host installiert sein, auf dem ich hinterher auch das Playbook ausführe - **nicht** auf den Systemen die ich konfigurieren will!

Lösung

Damit wird das ganze schon viel übersichtlicher! Wir brauchen: * eine Variablendatei `vars/role.yml`:

```
mysql_root_password: Sup3rS3cur3
mysql_databases:
  - name: example_db
    encoding: latin1
    collation: latin1_general_ci
mysql_users:
  - name: example_user
    host: "%"
    password: alm0sts3cure
    priv: "example_db.*:ALL"
```

- ein Playbook `role.yml`

```
---
- name: install and configure mysql the elegant way
  hosts: server
  become: yes
  vars_files:
    - vars/role.yml
```

```
  roles:
    - { role: geerlingguy.mysql }
```

. . . und das ist alles!

Wer beides ausprobieren will, muss zwischen den beiden Varianten allerdings auf dem server die Datenbank stoppen, deinstallieren, und das Datenverzeichnis /var/lib/mysql entfernen.