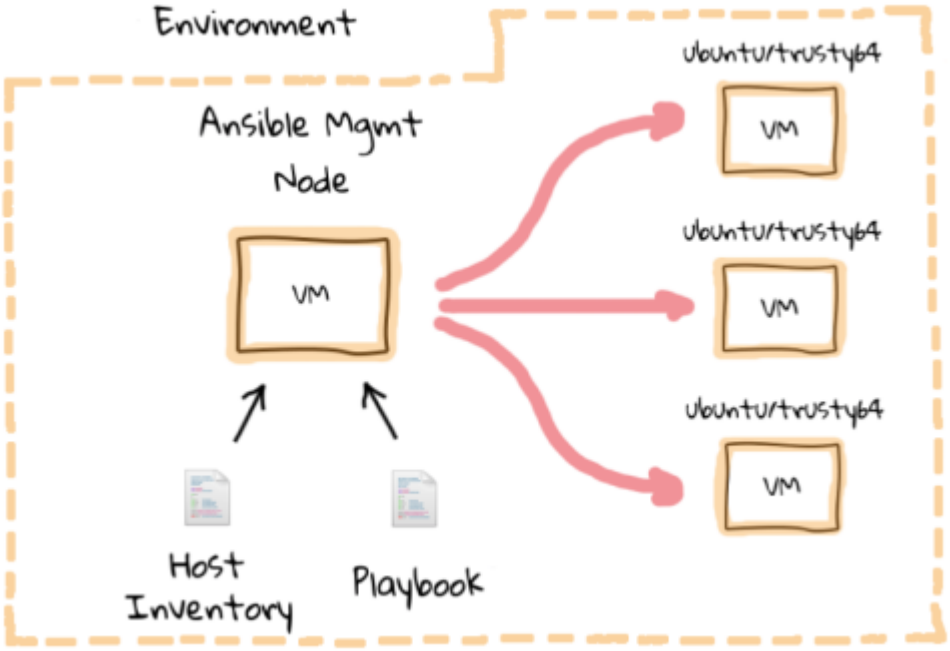
Ansible

Ansible by default manages machines over the SSH protocol.

Jeder Admin kennt sie: Sich immer wiederholende Arbeiten. Gerade in Zeiten von *(vielen)* virtuellen Maschinen, ggf. in der Cloud, oder geclusterten Anwendungen häufen sich diese Aufgaben und bei Änderungen am Setup muss sichergestellt werden, dass diese Änderungen auch auf allen betroffenen Maschinen umgesetzt werden.

Mit Ansible steht dem Admin ein junges und mächtiges Tool zur Verfügung, welches genau auf diese Arbeiten abzielt;



Im Gegensatz zu anderen Lösungen wie z. B. Puppet geht Ansible aber einen anderen Weg: Es beschreibt, "*wie*" man zu dem gewünschten Zustand kommt und nicht, "*was*" das Ziel sein soll. Für den Admin bedeutet dies, dass er seine bisher verwendeten, vielleicht sogar in ein Script gegossenen, Befehle in Ansible übersetzt, diese in eigenen **Playbooks** auf dem **Management Node** zusammenfasst - und so von den vielen Möglichkeiten der zahlreichen Module profitiert.

Die Einstiegshürde ist somit gering, kann auch neben bisherigen Lösungen verwendet werden und ist aufgrund der Verwendung von SSH als Transportweg nicht an die Installation auf den zu verwaltenden Maschinen gebunden.

Weitere nützliche Informationen:

- Das Ansible Funktionsprinzip anhand eines Videos und Grafiken erklärt.

Installation auf CentOS / RHEL7

Installiert, wird Ansible anschliessend mit einem ganz einfachen yum install auftruf; Da die benötigten

RPMs über das EPEL 6, 7, repositorie verfügbar sind. → (EPEL Repository konfigurieren.)

```
# yum install ansible
```

Erste Schritte mit Ansible

In den ersten Schritten, wird Ansible grundlegend Konfiguriert und erste Test-Kommandos abgesetzt!
Wichtig: Damit Ansible bei vorkonfigurierter SSH-Key-Authentifizierung richtig funktionieren kann, muss eine **SSH Verbindung** vom Ansibe-Host zu allen anzusprechenden Servern ohne Passwort möglich sein! Darum ist es zwingend, dass wir:

(Unter 'Key Auth.' in 'Putty Einstellungen' –> Agent-Forewarding aktivieren!)

Ansible-master Host Konfiguration

In der Datei `/etc/ansible/hosts`, werden alle **Hosts** definiert, welche durch Ansible angesprochen werden sollen. Diese können zu oberst untergruppiert angegeben werden, oder anschliessend gruppiert in zwei `[..]`. Unten wurden beispielshalber drei Gruppen mit verschiedenen Servern erstellt.

```
# vim /etc/ansible/hosts
```

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

[proxy]
192.168.1.6  # Odroid Proxy

[redhat]
192.168.1.10  # Wiki
192.168.1.14  # xweb
192.168.1.22  # plexdash.com

[debian]
192.168.1.23  # Plex-server
```

```
192.168.1.24  # Nextcloud_v2

...
```

Erste Ansible Test-Kommandos

Um erst einmal zu überprüfen, ob unsere Server überhaupt durch Ansible angesprochen werden können, kann man diese nun einzeln, oder auch als Gruppe an-pingen.

```
Pingen eines einzelnen Servers:
# ansible 192.168.1.14 -m ping

Pingen einer vorher definierten Gruppe:
# ansible redhat -m ping

Anpingen von mehreren Gruppen:
# ansible redhat,proxy -m ping
```

System Aktualisierung mit Ansible via Console

Etwas sehr nützliches, ist anschliessend auch die Möglichkeit, direkt Shell Kommandos über das eingebaute Shell Modul von Ansible, Kommandos parallel auf allen Hosts zur selben Zeit auszuführen. Hierbei das Beispiel, einer System Aktualisierung mit Ansible, für alle Systeme aus der Gruppe redhat:

```
# ansible redhat -s -m shell -a 'yum update -y'
```

- **-s** → Führt den abgesetzten Ansible Command auf den Zielsystemen als user **root** aus!
- **-m** → Modul angabe.. → `Hier das Modul “Shell”`!
- **-a** → Sind die Argumente für das Modul! → `Hier das Kommando yum update -y`.

Weitere Tipps, für einen einfachen Neueinstieg in Ansible, findet man unter folgendem Link: http://docs.ansible.com/ansible/intro_getting_started.html

---

Playbooks

Im folgenden Kapitel, wird nun das gesamte Thema der Ansible Playbooks einmal genauer angeschaut. *Jedoch zuerst, was sind eigentlich Playbooks?* Playbooks, sind im Grunde vorkonfigurierte und festgelegte SOLL-Zustände für spätere Zielsysteme. Diese werden in der Ansible eigenen sogenannten `configuration,deployment and orchestration language` definiert. Einfach ausgedrückt, kann man mit einem Ansible Playbook ein frisch installiertes System voll-automatisiert nach seinen Wünschen konfigurieren lassen.

Wichtig: Bevor später Playbook ausgeführt werden können, sollte man sicherstellen, dass man sich auf der `master-ansible-machine` via **SSH** und aktiven `agend forewarding` als *normalen User* **(nicht root!)** eingeloggt hat.

Falls überprüft werden soll, ob man auf den geplanten Zielsystemen auch wirklich root-Zugriffe hat; kann man dies mit folgendem Befehl überprüfen:

```
[rebermi@vstif2 ~]$ ansible all -s -m shell -a id

vstif1.pnet.ch | SUCCESS | rc=0 >>
uid=0(root) gid=0(root) groups=0(root)
```

Der Aufbau von einfachen Playbooks

Playbooks are expressed in YAML format (see YAML Syntax) and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.

Each playbook is composed of one or more ‘plays’ in a list.

The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module (see About Modules).

For starters, here’s a playbook that contains just one play:

```
---
- hosts: webservers
  remote_user: root
  vars:
    http_port: 80
    max_clients: 200
  tasks:
  - name: 'ensure apache is at the latest version'
    yum: name=httpd state=latest
  - name: 'write the apache config file'
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: 'ensure apache is running (and enable it at boot)'
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

When working with tasks that have really long parameters or modules that take many parameters, you can **break tasks items over multiple lines** like the example below; to improve the structure.

```
---
- hosts: webservers
  remote_user: root
  vars:
    http_port: 80
    max_clients: 200
  tasks:
  - name: 'ensure apache is at the latest version'
    yum:
      name: httpd
      state: latest
  - name: 'write the apache config file'
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - restart apache
  - name: 'ensure apache is running'
    service:
      name: httpd
      state: started
  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```

Playbooks can contain multiple plays. You may have a playbook that targets first the web servers, and then the database servers. For example:

```
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: 'ensure apache is at the latest version'
    yum: name=httpd state=latest
  - name: 'write the apache config file'
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf

- hosts: databases
  remote_user: root
```

```
  tasks:
  - name: 'ensure postgresql is at the latest version'
    yum: name=postgresql state=latest
  - name: 'ensure that postgresql is started'
    service: name=postgresql state=started
```

Achtung! Grössere Playbooks, werden dann wie im nachfolgenden Beispiel nicht nur noch in ein einziges YAML-File geschrieben, sondern in jeweilige Rollen und in einer eigenen Ordnungsstruktur zur besseren Verwaltung und Übersicht unterteilt. *(Nächstes Thema, nach der Syntax-Erläuterung)*

Erläuterung der Basis Syntax eines Playbooks

To simple explain the Syntax of the playbooks, let's first look at a basic playbook:

```
---
- hosts: debianwebservers

  tasks:
    - name: 'Installs nginx web server'
      apt: pkg=nginx state=installed update_cache=true
      notify:
        - start nginx

  handlers:
    - name: start nginx
      service: name=nginx state=started
```

Let's break this down in sections so we can understand how these files are built and what each piece means.

The file starts with:

```
---
```

This is a requirement for YAML to interpret the file as a proper document. YAML allows multiple “documents” to exist in one file, each separated by three minus-characters, but Ansible only wants one per file, so this should only be present at the top of the file.

YAML is very sensitive to white-space, and uses that to group different pieces of information

together. **You should use only spaces and not tabs and you must use consistent spacing for your file to be read correctly.** Items at the same level of indentation are considered sibling elements.

Items that begin with a `-` are considered list items. Items that have the format of `key: value` operate as hashes or dictionaries. That's pretty much all there is to basic YAML.

YAML documents basically define a hierarchical tree structure with the containing elements further to the left.

On the second line, we have this:

```
---
- hosts: debianwebservers
```

This is a list item in YAML as we learned above, but since it is at the left-most level, it is also an Ansible “play”. Plays are basically groups of tasks that are performed on a certain set of hosts to allow them to fulfill the function you want to assign to them. Each play must specify a host or group of hosts, as we do here.

Next, we have a set of tasks:

```
---
- hosts: debianwebservers

  tasks:
    - name: Installs nginx web server
      apt: pkg=nginx state=installed update_cache=true
      notify:
        - start nginx
```

At the top level, we have “**tasks:**” at the same level as “**hosts:**”. This contains a list (because it starts with a “**-**”) which contains key-value pairs.

The first one, “**name**”, is more of a description than a name. You can call this whatever you would like.

The next key is “**apt**”. This is a reference to an Ansible module, just like when we use the ansible command and type something like:

```
# ansible -m apt -a 'whatever' all
```

This module allows us to specify a package and the state that it should be in, which is “installed” in our case. The `update-cache=true` part tells our remote machine to update its package cache *(apt-*

get update) prior to installing the software.

The “**notify**” item contains a list with one item, which is called “start nginx”. This is not an internal Ansible command, it is a reference to a handler, which can perform certain functions when it is called from within a task. *We will define the “start nginx” handler below.*

```
---
- hosts: debianwebservers

  tasks:
    - name: Installs nginx web server
      apt: pkg=nginx state=installed update_cache=true
      notify:
        - start nginx

  handlers:
    - name: start nginx
      service: name=nginx state=started
```

The “**handlers**” section **exists at the same level** as the “**hosts**” and “**tasks**”. Handlers are just like tasks, but they only run when they have been told by a task that changes have occurred on the client system.

For instance, we have a handler here that starts the Nginx service after the package is installed. The handler is not called unless the “Installs nginx web server” task results in changes to the system, meaning that the package had to be installed and wasn't already there.

Eigenes strukturiertes Playbook erstellen

Als grösseres Beispiel, wird hier nun ein neues **Playbook** mit diversen Rollen zum automatischen konfigurieren eines **LAMP Stack mit Ansible** erstellt. Der Aufbau des neuen Playbooks, habe ich zur besseren Verdeutlichung in Schritte unterteilt.

Schritt 1 - Erstellen des neuen Playbook-Ordners

Zur besseren Übersicht, sollte für jedes neue Ansible Playbook, auch einen neune Ordner mit dem jeweiligen Namen passend zum Inhalt des Playbooks erstellt werden.

```
# mkdir /home/rebermi/demo-playbook-lamp-rhel7
# cd ~/demo-playbook-lamp-rhel7
```

Für unser Beispiel, werden wir in einem separaten Ordner unsere Variablen definieren, drei unterschiedliche Rollen erstellen und Konfigurationstemplates zu einzelnen Pakete vordefinieren. Unsere fertige Struktur, sollte nach unserer Arbeit in etwa so aussehen:

```
.
├── deploy-lamp.yml
├── group_vars
│   ├── all
│   └── dbservers
├── hosts
└── roles
    ├── common
    │   ├── handlers
    │   │   └── main.yml
    │   ├── tasks
    │   │   └── main.yml
    │   └── templates
    │       └── ntp.conf.j2
    ├── db
    │   ├── handlers
    │   │   └── main.yml
    │   ├── tasks
    │   │   ├── main.yml
    │   │   └── secure_installation.yml
    │   └── templates
    │       └── my.cnf.j2
    └── web
        ├── tasks
        │   └── main.yml
        └── templates
            └── index.php.j2

13 directories, 13 files
```

Schritt 2 - Let's start! - Anlegen der Haupt-Playbook Files

Zu beginn, wird nun erst einmal die Hauptdatei des Playbooks angelegt, welche dann später all unsere erstellten Rollen zusammenfasst und den richtigen Servern zuweist. Wir nennen diese Datei hier `deploy-lamp.yml` und füllen sie wie folgt ab!

```
# vim /home/rebermi/demo-playbook-lamp-rhel7/deploy-lamp.yml
```

```
---
# Hauptdatei, welche alle Rollen zusammenfasst.
- name: Common configuration to all nodes
  hosts: all
  become: true
  become_user: root
  roles:
    - common

- name: Configure and deploy the webserver and the code.
```

```
  hosts: webservers
  become: true
  become_user: root
  roles:
    - web

- name: Deploy MySQL and configure the database
  hosts: dbservers
  become: true
  become_user: root
  roles:
    - db
```

Als nächstes müssen wir hier für unser kleines Projekt, noch eine eigene `hosts` Datei erstellen. Dies machen wir, da wir die Hosts “webservers” und “dbservers” nicht in unserem System fix unter /etc/ansible/hosts definieren wollen; sondern diese nur einmal in diesem Playbook gebrauchen wollen. Darum erstellen wir unsere Hostsdatei wie folgt:

```
# vim /home/rebermi/demo-playbook-lamp-rhel7/hosts
```

```
[webservers]
vstif1.pnet.ch

[dbservers]
vstif1.pnet.ch
```

Schritt 3 - Erstellen der Rollen Struktur für unser Playbook

Damit wir anschliessend unsere Rollen planen können, muss zuerst eine **korrekte Ordnungsstruktur** erstellt werden.

Beginnen wir darum mit dem erstellen der Ordner:

```
# mkdir -p roles/{common,db,web}/{handlers,tasks,templates}
```

Dieses Kommando legt uns nun folgende Ordnungsstruktur an:

```
roles/
├── common
│   ├── handlers
│   ├── tasks
│   └── templates
├── db
│   ├── handlers
```

```
│   ├── tasks
│   └── templates
└── web
    ├── handlers
    ├── tasks
    └── templates

12 directories, 0 files
```

Schritt 4 - Abfüllen der common Rolle

In der common-Rolle konfigurieren wir generell auf allen erwähnten Servern den NTP Dienst. Das dies überhaupt möglich ist, müssen wir jedoch zuerst noch die dementsprechenden Files in den dafür vorgesehenen Ordnern erstellen und befüllen.

Beginnen wir mit der Hauptdatei im tasks Ordner:

```
# vim roles/common/tasks/main.yml
```

```
---
# Installs the NTP Package (Deamon) if it not exists..
- name: Install ntp
  yum: name=ntp state=present
  tags: ntp

# Installs some common dependencies from the list below.
- name: Install common dependencies
  yum: name={{ item }} state=installed
  with_items:
   - libselinux-python
   - libsemanage-python
   - firewalld

# Copy the ntp.conf.j2 template vorm Ansible-master to the applicationserver
- name: Configure ntp file
  template: src=ntp.conf.j2 dest=/etc/ntp.conf
  tags: ntp
  notify: restart ntp  #(If configuration is changed by ansible, notify will
contact the Handler="restart ntp")

# This task starts and enables the ntp deamon
- name: Start the ntp service
  service: name=ntpd state=started enabled=yes
  tags: ntp
```

Als nächstes erstellen wir unser Konfigurationstemplate für den NTP-Deamon:

```
# vim roles/common/templates/ntp.conf.j2
```

```
# --- General settings ------------------

driftfile /var/lib/ntp/drift/ntp.drift  # for Linux jail run mode on SuSE
11.x

server {{ ntpserver }}


# --- IP filtering (Security) ------------------
#restrict default ignore         # close all


# local machine
#restrict 127.127.1.0    # unrestrict the local clock
restrict 127.0.0.1      # Loopback address
restrict ::1            # IPv6 loopback address
#restrict 10.1.100.181   # unrestrict the host itself
```

Zum Schluss erstellen wir noch den Handler zum restarten des NTP Deamons:

```
# vim roles/common/handlers/main.yml
```

```
---
- name: restart ntp
  service: name=ntpd state=restarted
```

Schritt 5 - Abfüllen der db Rolle

In der db-Rolle konfigurieren wir unsere Datenbank. Diese wird wie aus der Hauptdatei ersichtlich ist, nur auf jenen Servern installiert, welche sich in der `dbservers` Gruppe befinden.

Beginnen wir wieder mit der Hauptdatei im tasks Ordner:

```
# vim roles/db/tasks/main.yml
```

```
---
# Insalliert MariaDB, erstellt db-User, setzt Berechtigungen und führt die
Secure-Installation aus.

- name: Install MariaDB
```

```
  yum: name={{ item }} state=installed
  with_items:
   - mariadb-server
   - MySQL-python

- name: Configure SELinux to start mysql on any port
  seboolean: name=mysql_connect_any state=true persistent=yes

- name: Create Mysql configuration file
  template: src=my.cnf.j2 dest=/etc/my.cnf
  notify:
  - restart mariadb

# Erstellen & berechtigen der in der Config gebraucheten Files
- name: Create MariaDB log file
  file: path=/var/log/mysqld.log state=touch owner=mysql group=mysql
mode=0775

- name: Create MariaDB PID directory
  file: path=/var/run/mysqld state=directory owner=mysql group=mysql
mode=0775

- name: Start MariaDB Service
  service: name=mariadb
           state=started
           enabled=yes

- name: Start firewalld
  service: name=firewalld
           state=started
           enabled=yes

- name: insert firewalld rule
  firewalld: port={{ mysql_port }}/tcp permanent=true state=enabled
immediate=yes

- include: secure_installation.yml

- name: Create Application Database
  mysql_db: login_user=root login_password="{{ mysql_root_password }}"
name={{ dbname }} state=present

- name: Create Application DB User
  mysql_user: login_user=root login_password="{{ mysql_root_password }}"
name={{ dbuser }} password={{ upassword }} priv=*.*:ALL host='%'
state=present
```

Da hier nun auch noch ein zweites Task-file für die Secure_Installation eingebunden wird, müssen wir dies selbstverständlich auch noch unter erstellen!

```
# vim roles/db/tasks/secure_installation.yml
```

```
---
#mysql Secure Installation:
- name: delete anonymous MySQL server user for {{ ansible_nodename }}
  mysql_user: login_user=root
              login_password='{{ mysql_root_password }}'
              check_implicit_admin=yes
              user=""
              host={{ item }}
              state="absent"
  with_items:
    - ""
    - "{{ ansible_nodename }}"
    - localhost

- name: Change root user password on first run
  mysql_user: login_user=root
              login_password="{{ mysql_root_password }}"
              check_implicit_admin=yes
              name=root
              password={{ mysql_root_password }}
              priv=*.*:ALL,GRANT
              host={{ item }}
  with_items:
    - 127.0.0.1
    - ::1
    - localhost

- name: remove the MySQL test database
  action: mysql_db login_user=root login_password="{{ mysql_root_password
}}" db=test state=absent
```

Als nächstes erstellen wir das Konfigurationstemplate für den MariaDB-Deamon:

```
# vim roles/db/templates/my.cnf.j2
```

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0
port={{ mysql_port }}
```

```
[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Zum Schluss wird auch hier wieder der Handler zum restarten des MariaDB Deamons erstellt:

```
# vim roles/db/handlers/main.yml
```

```
---
- name: restart mariadb
  service: name=mariadb state=restarted
```

Schritt 6 - Abfüllen der web Rolle

In der web-Rolle konfigurieren wir unseren Apache 2.4 Webserver. Dieser wird hingegen zum db-Server, nur auf jenen Servern installiert, welche sich in der `webservers` Gruppe befinden.

Beginnen wir wieder mit der Hauptdatei im tasks Ordner:

```
# vim roles/web/tasks/main.yml
```

```
---
# Hier wird der Webserver & PHP Installiert
- name: Install httpd and php
  yum: name={{ item }} state=present
  with_items:
   - httpd
   - php
   - php-mysql

- name: Install web role specific dependencies
  yum: name={{ item }} state=installed
  with_items:
   - git

- name: Start firewalld
  service: name=firewalld state=started enabled=yes

- name: insert firewalld rule for httpd
  firewalld: port={{ httpd_port }}/tcp permanent=true state=enabled
immediate=yes

- name: http service state
  service: name=httpd state=started enabled=yes
```

```
- name: Configure SELinux to allow httpd to connect to remote database
  seboolean: name=httpd_can_network_connect_db state=true persistent=yes



# Falls gewünst, kann mit folgenden zwei Zeilen auch ein CMS von einem
GitHub Repository heruntergeladen und in das Webverzeichniss verschoben
werden.
#- name: Copy the code from repository
#  git: repo={{ repository }} dest=/var/www/html/

# Hier wird die Test index.php erstellt:
- name: Creates the index.php file
  template: src=index.php.j2 dest=/var/www/html/index.php
```

Zum Schluss erstellen wir noch das Template für unsere index.php:

```
# vim roles/web/templates/index.php.j2
```

```
<html>
 <head>
  <title>Application-Server Installation, with Ansible</title>
 </head>
 <body>
 </br>
  <a href=http://{{ ansible_default_ipv4.address }}/index.html>Homepage</a>
 </br>
<?php
 Print "Hello, World! I am a web server configured using Ansible and I am :
";
 echo exec('hostname');
 Print  "</BR>";
echo  "List of Databases: </BR>";
        {% for host in groups['dbservers'] %}
                $link = mysqli_connect('{{
hostvars[host].ansible_default_ipv4.address }}', '{{ hostvars[host].dbuser
}}', '{{ hostvars[host].upassword }}') or die(mysqli_connect_error($link));
        {% endfor %}
        $res = mysqli_query($link, "SHOW DATABASES;");
        while ($row = mysqli_fetch_assoc($res)) {
                echo $row['Database'] . "\n";
        }
?>
</body>
</html>
```

Schritt 7 - Erstellen der zentralen Variablensammlung

Zum Abschluss unseres selbst-erstellten Playbooks, müssen wir nun natürlich noch sämtliche (In den Rollen und Konfigurationen verwendeten) Variablen definieren und zuweisen. Dies machen wir in dem Verzeichnis group_vars.

Da dieses noch nicht existiert erstellen wir dies zuerst:

```
# mkdir group_vars
```

Wichtig: Die hier definierten und erstellten Variablen-Dateien, müssen mit dem Namen der Hostgruppen übereinstimmen, damit diese für den jeweiligen Play verwendet werden können! Alternativ, erstellt man eine datei “all” welche dann Variablen für alle Plays definieren kann.

Erstellen der Variablensammlung für alle Hosts:

```
# vim group_vars/all
```

```
# Variables listed here are applicable to all host groups
httpd_port: 80
ntpserver: ch.pool.ntp.org
repository: https://github.com/bennojoy/mywebapp.git
```

Erstellen der Variablenzusammenstellung für den DB-Server:

```
# vim group_vars/dbservers
```

```
# Variabenl für die dbservers Gruppe

mysqlservice: mysqld
mysql_port: 3306
dbuser: michael
dbname: application-server-db1
upassword: abc123!M

mysql_root_password: theS3creTR00TpW!
```

Glückwunsch! Nun ist das erste Playbook fertigestellt!

Ausführen von Ansible Playbooks

Ansible Playbooks, können einfach so gestartet werden, oder auch mit der Angabe einer mitgelieferten hosts datei. **Um das Playbook vom oberen Beispiel auszuführen, sollte man die Variante 2 benützen!**

Variante 1 - Playbook ohne hosts Angabe ausführen

In dieser Variante, gellten alle vordefinierten Gruppen und zuordnungen unter `/etc/ansible/hosts` für das aktuelle Playbook!

```
# cd /root/ansible-playbooks/MY_AWESOME_PLAYBOOK_FOLDER
# ansible-playbook playbook.yml
```

Variante 2 - Playbook mit hosts Angabe ausführen

Bei dieser Variante, wird die System-eigene hosts Datei mit der vom aktuellen Ansible Playbook enthaltenen Datei übersteuert. Das heisst, es gelten nur diese Definitionen und Host-Zuordnungen aus dem Playbook!

```
# cd /root/ansible-playbooks/MY_AWESOME_PLAYBOOK_FOLDER
# ansible-playbook -i hosts playbook.yml
```

Weitere Themen zu Playbooks

Hier werden nützliche Links mit Tipps zum erstellen von eigenen Playbooks, sowie schon fertig erstellte Playbooks welche heruntergeladen angepasst und gleich verwendet werden gesammelt.

Fertige Playbooks zum Download

Unter dem unten aufgeführten Link, werden von Ansible unteranderem bereits fertige Playbooks, zu folgenden Themen zum Download angeboten: → `jboss-standolone`, `lamp_simple_rhel7`, `mongodb`, `tomcat-standalone`, `wordpress-nginx-rhel7` und viele mehr.

- GitHub Link → https://github.com/ansible/ansible-examples

Weitere nützliche Links

- http://docs.ansible.com/ansible/playbooks.html
- https://coderwall.com/p/6zm8rq/how-to-create-a-lamp-stack-with-ansible
- https://nsrc.org/workshops/2015/rwnog/raw-attachment/wiki/Track2Agenda/first-playbook.htm
- https://cloudacademy.com/blog/building-ansible-playbooks-step-by-step/

Ansible Module

Mit den Ansible Module, sagen wir Ansible, wie es mit der jeweiligen Situation, sprich wie es das angefügte Argument verarbeiten soll. Weiter können Module dazu gebraucht werden, um automatische Benachrichtigungen von abgeschlossenen Ansible-Tasks an den Auftraggeber oder auch nur an den master-ansible-host zurückzusenden.

Übersicht über alle Module

Sämtliche Ansible **Module**, werden für eine bessere Übersicht in folgende **verschiedene Kategorien** unterteilt. Mit einem klick, auf die jeweilige Modul Kategorie, wird man anschliessend auf dem Ansible Module-Index weitergeleitet und kann die einzelnen dazugehörigen Module näher betrachten.

- Cloud Modules
- Clustering Modules
- Commands Modules
- Crypto Modules
- Database Modules
- Files Modules
- Identity Modules
- Inventory Modules
- Messaging Modules
- Monitoring Modules
- Net Tools Modules
- Network Modules
- Notification Modules
- Packaging Modules
- Remote Management Modules
- Source Control Modules
- Storage Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules

Eigene Module erstellen

Weiteres

- http://docs.ansible.com/ansible/intro_getting_started.html

- https://serversforhackers.com/an-ansible-tutorial
- https://www.linode.com/docs/applications/configuration-management/learn-how-to-install-ansible-and-run-playbooks

---

Last update: **2017/07/18 17:14**