Hochschule für Technik und Wirtschaft Berlin
Internationaler Studiengang Medieninformatik - Master

Switching to Ansible

Moving a server from Docker-Compose

Florian Reitz

Independent Coursework

Betreuer: Prof. Dr.-Ing. David Strippgen

Berlin, 13.03.2022

Summary

German

Diese Arbeit beschäftigt sich mit den Grundlagen von Ansible und der Portierung einer Serverstruktur von Docker-Compose zu Ansible. Dabei wurde zuerst auf die Installation und Einrichtung von Ansible ein Blick geworfen. Danach wurde ein Webserver ohne Docker, zusammen mit einer Webseite automatisiert ausgerollt. Zuletzt wird ein genauerer Blick auf Ansible und Docker geworfen.

English

This work deals with the basics of Ansible and the portation of a server from Docker-Compose to Ansible. For this, a quick look is taken on the Installation and Configuration of Ansible and afterwards on the automatic deployment of a web server with a website without Docker. Lastly, the server stack is moved from Docker-Compose, and the Docker module of Ansible is further examined.

Table of contents

List of Figures

1 Introduction

Ansible is an agent-less system that needs no special software to run on the remote node for automating tasks. It uses so-called modules, which are libraries, to extend its functions. For example, in this work, the docker-module is used to a great extend. All configuration files are written in the YAML and can be called over the command line. By default, Ansible uses OpenSSH to connect to the remote system.

This work aims to inspect Ansible and move a server from Docker-Compose to Ansible. To better observe non-Docker-related functions of Ansible, the setup of a web server, with the deployment of a website, was automated. Lastly, a comparison is made, and current alternatives are discussed.

The complete Code for this work can be found at `https://github.com/troppes/playbooks`.

Some files are encrypted to keep passwords safe. For these files, a .dist counterpart is inside the repository that displays the structure of the other file.

2

Installation and Configuration

2.1 Controller

The Controller-System takes the scripts, so-called Playbooks 3 and executes them on the other systems. It is the only system that needs to have ansible installed.

2.1.1 Windows

In the foreseeable future, it is not possible to install Ansible on Windows Systems when disregarding the WSL[1]. One of the biggest reasons for this is the heavy reliance on the $fork()$-behavior, which is not implemented in Windows. [1]

2.1.2 Linux

For the Linux installation, the WSL2 with Ubuntu was used. To install the application, it was first necessary to install a custom PPA[2]. Then the Controller could be installed through APT[3] [2]

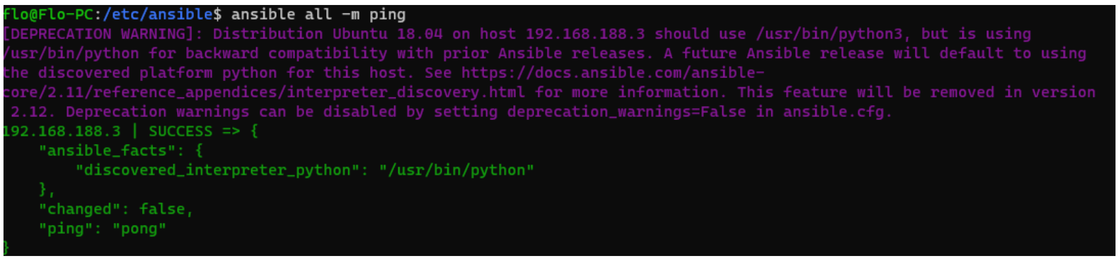
2.2 Setting up the first machine

To add a machine access has to be established using an ssh-key. For this, it was necessary to generate a private-public key-pair and then install the public key on the server. Afterwards, it was possible to log in without using a password.

The server IP can be set in the Ansible host config file '/etc/ansible/hosts' to add the server to Ansible. A ping can be sent out to all host systems in order to test if the connection works.

When pinging the server, a deprecation message appeared, to see in figure 2.1. Since the server had python3 installed, it was unclear where this message came from. The problem was that when given a choice, Ansible was using the older python version. A variable in the Ansible-Configuration must be changed to use python3 automatically which fixes this behavior. The usage of python3 will be the default behavior in the next release from Ansible. [3]

[1] Windows Linux Subsystem
[2] Personal Package Archive. A source from which software can be installed.
[3] Advanced Packaging Tool. The package manager of Ubuntu.



```
flo@Flo-PC:/etc/ansible$ ansible all -m ping
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host 192.168.188.3 should use /usr/bin/python3, but is using
/usr/bin/python for backward compatibility with prior Ansible releases. A future Ansible release will default to using
the discovered platform python for this host. See https://docs.ansible.com/ansible-
core/2.11/reference_appendices/interpreter_discovery.html for more information. This feature will be removed in version
 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
192.168.188.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Abb. 2.1. Ansible deprecation message

3

Playbooks

Playbooks are YAML-Files[1], that are used to execute a list of commands on the remote system and are Ansible's primary working method.

3.1 First Playbook

```
---
— name: My task
  hosts: all
  remote_user: flo
  tasks:
      - name: Leaving a mark
        command: "touch /tmp/ansible_was_here"
      - name: Providing feedback
        command: echo "Playbooks works"
        register: feedback
      - debug: msg="{{feedback.stdout}}"
```

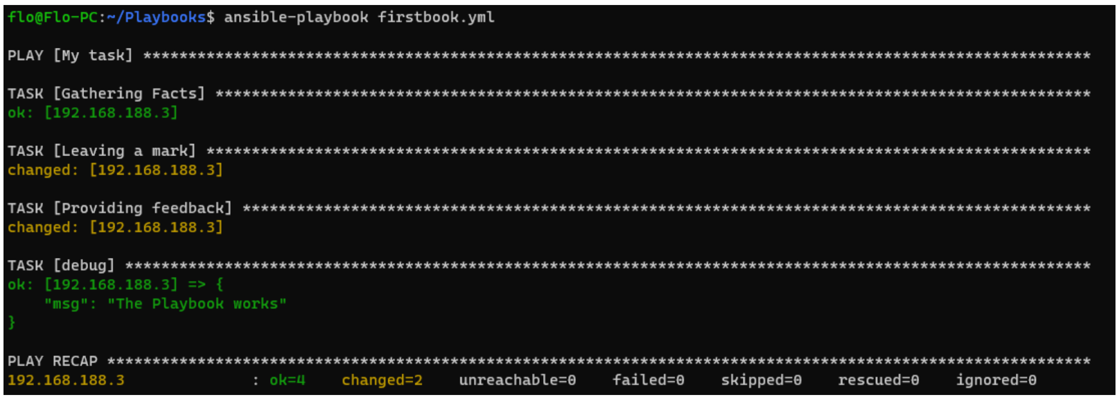
Listing 3.1. First Playbook

Playbooks always start with three dashes. To check if the playbook's syntax is correct, Ansible has a built-in syntax checker and a linter if customs styles are needed.

After the dashes, the name of the so-called play, a list of rules, is defined. In this example, it is "My task."Then some configurations like the user and hosts can be set. Below it are the tasks that are called during execution. The first playbook creates a file and then writes on the command line. Normally the output of playbooks is not shown on the command shell, so it is necessary to use a register to write all output into a variable. Afterwards, the debug output was used to get the values of the variable. The execution can be seen in Figure 3.1.

When looking at the output in Figure 3.1, it can be seen that sometimes 'OK' is returned and sometimes 'Changed.' 'Changed' is used if the module, in this case 'command', considers something meaningful changed. The definition of meaningful is left to the module creator. [4]

[1] Yet Another Macro Language



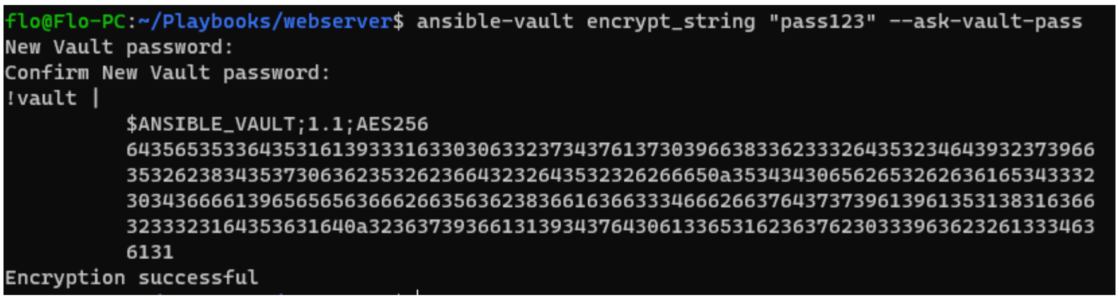
```
flo@Flo-PC:~/Playbooks$ ansible-playbook firstbook.yml

PLAY [My task] ************************************************************************************************

TASK [Gathering Facts] ****************************************************************************************
ok: [192.168.188.3]

TASK [Leaving a mark] *****************************************************************************************
changed: [192.168.188.3]

TASK [Providing feedback] *************************************************************************************
changed: [192.168.188.3]

TASK [debug] **************************************************************************************************
ok: [192.168.188.3] => {
    "msg": "The Playbook works"
}

PLAY RECAP ****************************************************************************************************
192.168.188.3              : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Abb. 3.1. Output of the first Playbook

4 Encryption

Encryption is used in the project to hide all secrets needed for the creation of the server stack, such as the database passwords. In Ansible, there are two different ways of encrypting.

The first method is encrypting only the strings. Therefore, we use the Ansible Vault to encrypt the password, as seen in Figure 4.1 and store it into the host variables file.



```
flo@Flo-PC:~/Playbooks/webserver$ ansible-vault encrypt_string "pass123" --ask-vault-pass
New Vault password:
Confirm New Vault password:
!vault |
          $ANSIBLE_VAULT;1.1;AES256
          64356535336435316139333163303063323734376137303966383362333264353234643932373966
          3532623834353730636235326236643232643532326266650a353434306562653262636165343332
          30343666613965656563666266356362383661636633346662663764373739613961353138316366
          3233323164353631640a323637393661313934376430613365316236376230333963623261333463
          6131
Encryption successful
```

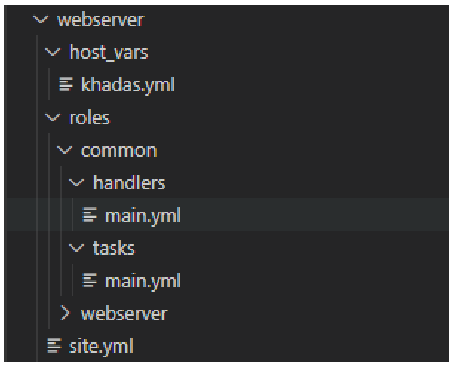
Abb. 4.1. Usage of the ansible-vault command

Since this file is automatically read when connecting to the specific host, the root password can be encrypted. After the file is encrypted, it is necessary to add the –ask-vault-pass parameter to the playbook call to decrypt the file. If no password is supplied or the parameter is not set, the command will terminate in error.

The second method is to pass in a complete file as an argument. This leads it to be fully encrypted. The command usage is similar, but the encrypt_string argument is dropped. When multiple files or strings are encrypted with the same password, they all get unlocked if the –ask-vault-pass parameter is used.

5 Playbook Structure

No actual structure was necessary for the first example since it only created one file. However, for the following more complex Playbooks, a structure is needed. For this, the recommended structure from Ansible is used. The structure of the Web server-Playbook can be seen in Figure 5.1.



```
webserver
  host_vars
    khadas.yml
  roles
    common
      handlers
        main.yml
      tasks
        main.yml
    webserver
  site.yml
```

Abb. 5.1. Folder Structure of the Web server Playbook

5.1 Host and Group Variables

In the Host Variables folder, system-specific files are stored, like the root password for the system. The file name must match either the hostname or an alias.

Additionally, a folder called group_vars can be created to manage hosts' groups. If global variables are needed, it is possible to create a file called all.yml 7.6, which can be read by every system.

5.2 Roles

Roles are collections of tasks in Ansible. For example, one role updates the server, another deploys a website, and a third remove the website. In general, a role is everything that is needed to handle this part of the process.

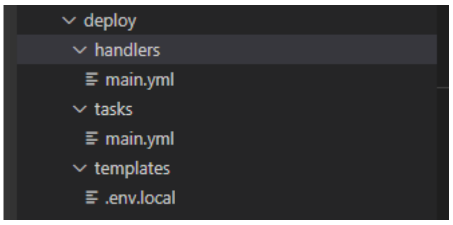


Abb. 5.2. Example of the folder structure of a role

5.2.1 Tasks

The tasks folder contains a main.yml executed when the Role is called. The modules can be called to make changes in the remote system, for example, the command module in the first playbook. It is best practice to give every module-call (task) a separate name.

If the main.yml gets too big, it may be split up. The other files need to be included in the main.yml afterwards in order to be called.

5.2.2 Handlers

In the Handler folder, the handlers for the Role are defined. Handlers are tasks called upon after a task modifies something and returns the status changed. An example of this can be found for the Nextcloud-Role 7.5, where if the container is recreated an SVG-extension needs to be installed.

5.2.3 Template

The template folder is used to store files that need to be used during the tasks. It enables the use of the template module to copy files.

Furthermore, it allows the files stored in the folder to use variables 5.1 from the playbooks inside them. This way, secrets can be dynamically placed before uploading the file.

If only the copy mechanism is needed instead of creating a template folder, a files folder can be used with the copy module. Since there was no noticeable performance hit when using a template, it was used for greater flexibility.

5.2.4 Variables and Defaults

When a role needs special variables, the vars folder can be used. Defaults can be set inside the defaults folder, which is overwritten when the same variable is defined in vars. Variables get overwritten the more local they are. So if a variable is written in a role, it will overwrite the host and group variables.

Neither was used here, since not many variables were needed. The variables used were global variables, since storing them in one file turned them more readable.

5.2.5 Meta and Library

The meta-folder is used to set metadata, like the author, and role dependencies. By doing this, other roles can be called from inside a role.

The library folder is used when writing own modules for particular tasks. These can be stored there to call them inside the role. The same is possible in the top-level directory if the modules shall be available to all roles.

Since these playbooks will not be published and own custom modules were not created, both folders were not used during this work.

5.3 YAML-Files in the main folder

In the Main folder, there can be multiple YAML-Files. Those are the files that are called when running the playbook and contain the following configuration: to which systems the playbook deploys, which roles are called and which user is used to connect.

6

Web Server Playbook

The web server playbook updates the system and installs NGINX and Node. Afterwards, it pulls the repository for the web page and then builds and deploys it.

6.1 Structure

This playbook has four different roles. The first updates the system, the second pulls the website from Github and builds it. The third installs and configures NGINX. Furthermore, the last role removes all the installed software and the website.

6.1.1 Update-Role

The first role has three tasks. First, it updates the cache with the apt-module from Ansible.

```
- name: Update Cache
  apt:
    update_cache: true

- name: Update all packages
  apt:
    name: "*"
    state: latest
```

Listing 6.1. The update task

Then it updates all installed packages, as seen in Listing 6.1. Lastly, it upgrades the operating system. The presented commands want to receive a state argument, which is often used by Ansible modules. It shows the wanted state of the operation. In this case, the latest. After updating the update-cache, Ansible will compare the currently installed version of the apps with the latest version found in the update-cache and update it, if needed.

6.1.2 Deploy-Role

The second role first adds the Nodesource repository to install a version of Node that is newer than the current one on the official repository. This had to be done, since the website does not support the old version of Node. Afterwards, it installs Node and Yarn and pulls the latest version of the website.

The git-module can automatically pull repositories and needs only the link to the repository and the drive's destination as base arguments. The default linter used to ensure the correct usage demands a version argument as well. Hence it was necessary to create a Version on GitHub to satisfy it. After the building and deployment process, a call is made on a handler to delete the downloaded repository. Since handlers are always called at the end, it can be ensured that the repository is only deleted after the run has ended.

Then the Configuration File with the needed API-Keys was put onto the system via the template command.

Afterwards a yarn install is called to install the needed node libraries to build the project. Since the community-built yarn module did not support building, it was necessary to use the command module from Ansible. The linter was disabled on this step with a comment, as seen in Listing 6.2. This is because the operation needs to run every time and the linter demands that a when-argument is added to make sure the call is necessary. The command module itself needs the cmd-argument for the command and the chdir for the working directory to function.

```yaml
- name: Git checkout
  git:
    repo: 'https://github.com/troppes/personal-website.git'
    dest: /home/flo/web
    version: v0.8.0
  notify: Cleanup

- name: Deploy Custom configs
  template:
    mode: 0777
    src: .env.local
    dest: /home/flo/web/.env.local
 [...]
- name: Yarn Build  # noqa no-changed-when
  command:
    cmd: yarn build
    chdir: /home/flo/web
```

Listing 6.2. The update task

Lastly, the built web page is moved. After the tasks are finished, the handler is called, removing the not-needed folders.

6.1.3 Web server-Role

The built website now has to be made available to the general public. To do this, first, NGINX needs to be installed. Afterwards, the configuration file for the webserver is copied to the host. Lastly, the symlink to make the page available is modified. When either of the two steps before are changing something, a handler restarts NGINX.

6.1.4 Removal-Role

This role removes all files, the web server, and the software used to build the website. First, it checks if the NGINX-Service still exists, and based on that, NGINX is stopped and removed. For this, the when-argument is used, as seen in Listing 6.3. The service_facts module scans all the available services and writes them to the services variable. The SStop NGINXTask is executed if NGINX is found inside the variable. Next, NodeJS is removed, and lastly, the folder where the website was located is deleted.

```
- name: Find Sevices
  service_facts:

- name: Stop Nginx
  service:
    name: nginx
    state: stopped
  when: "'nginx' in services"
```

Listing 6.3. Usage of the when-argument

7

Docker Playbook

7.1 Current Setup

Currently, all my services are orchestrated by docker-compose. The goal of this part is to move the setup, seen in Figure 7.1, entirely to Ansible.

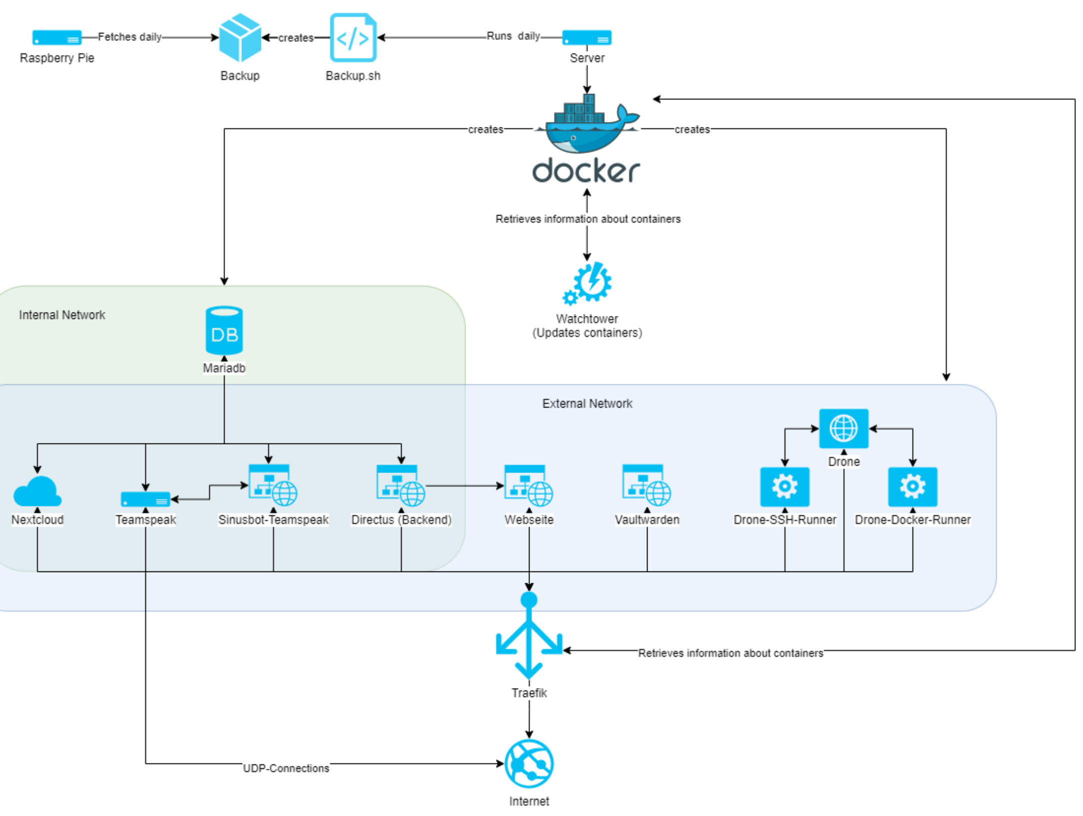


Abb. 7.1. Server Stack

The current setup has two Networks, one with internet access and one without. Only the database and services connected directly to the database reside in the network without internet access. The only service in the default docker network is Watchtower, which automatically updates containers. In the second network, all services which need Internet access reside. Most of them run through the Edge Router Traefik, used as a reverse proxy. Since UDP is not well supported in Traefik yet, Teamspeak needs to bypass some traffic.

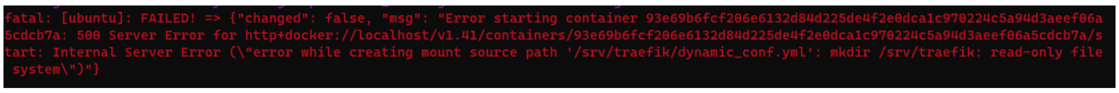
Teamspeak contains a Bot, which has an internal connection to the Teamspeak Container.

Directus supplies the website with content through a REST API. This call is external, and the connection only demonstrates that data is sent, but the website uses the router to request the information.

Lastly, the CI/CD-Service Drone uses sub-containers to execute pipelines. Here two separate containers are used. The first is used to run docker commands, like building containers, and the second container runs SSH jobs.

The backup structure is further explained in 7.7.1.

Before working on the live server, an Ubuntu-VM was used to test changes. The VM was created in Hyper-V since WSL uses some features of Hyper-V. Nevertheless, by default, it does not use the same network. A new switch needed to be created to allow the WSL to communicate with the VM, which allowed external access. While setting up the VM, the setup offered to install Docker directly, which initially seemed convenient. Nonetheless, it turned out to be a mistake, since the installed Version produces an error, Fig. 7.2, during deployment. The error made it look like the problem was in the file rights, which led to long debugging until reinstalling. After that, the error disappeared.



```
fatal: [ubuntu]: FAILED! => {"changed": false, "msg": "Error starting container 93e69b6fcf206e6132d84d225de4f2e0dca1c970224c5a94d3aeef06a
5cdcb7a: 500 Server Error for http+docker://localhost/v1.41/containers/93e69b6fcf206e6132d84d225de4f2e0dca1c970224c5a94d3aeef06a5cdcb7a/s
tart: Internal Server Error (\"error while creating mount source path '/srv/traefik/dynamic_conf.yml': mkdir /srv/traefik: read-only file
 system\")"}
```

Abb. 7.2. Docker Error during Deployment

7.2 First steps in Ansible Docker

Docker Ansible does not work directly on the server, as it requires the Python Docker library. The main server had it already installed since Docker-Compose is built on the same library, but the package ”python3-docker”needed to be installed on the test server. After that, the commands started to work, and it was possible to pull and deploy a website as a test.

7.3 Docker-Module

The Docker-Module works similarly to an entry in the docker-compose.

```yaml
---
- name: Deploy Traefik
  community.docker.docker_container:
    image: traefik:latest
    name: traefik
    state: started
    restart_policy: unless-stopped
    purge_networks: true  # Removes Container from all Networks
    network_mode: default
    networks:
      - name: "traefik"
    ports:
      - "80:80"
      [...]
    volumes:
      - /etc/localtime:/etc/localtime:ro
      [...]
    security_opts:
      - no-new-privileges:true
```

Listing 7.1. Docker Module

Some significant changes are the purge_networks parameter, which removes the container from all networks, except those named in the network parameter. Another change is that a state needs to be set instead of assuming that the container needs to be started. This is needed to stop containers, which is done with an extra argument in the docker-compose command line call.

7.4 Roles

This chapter briefly reviews some of the roles with unique mechanisms.

7.4.1 Common

There were two modules to create Docker networks. One from Ansible directly and one from the community. The common role sets up the internal and external network. The development on the Ansible one has stopped, since the community module has more functions, like the creation of internal networks, which were needed for the database network.

```yaml
---
- name: Create Traefik External Network
  community.docker.docker_network:
    name: traefik
    internal: false
```

Listing 7.2. Network-Creation Taks

7.4.2 Traefik

I use the Reverse-Proxy Traefik to handle the incoming traffic. This proxy automatically routes the traffic to the right docker-container based on the accessed Domain and Sub-Domain.

The Windows hosts-File needed to be rewritten to redirect the traffic to the VM to work. Unfortunately, the Windows-Host-File does not support Wildcards, which lead to every Sub-Domain being manually written.

Two files needed to be transferred and an empty one created for Traefik to work. The first two files get copied by the template module.

The empty file gets then filled by Traefik with the Lets Encrypt certificate. In order to create the empty file, the file module was used with the touched-state. This led to unwanted behavior, since touch updates the access time and modification time every time the command is run. The leads to a changed status every time. However, for this use-case, the timestamps are not relevant. So the access_time and modification_time needed to be preserved, which can be seen in Listing 7.3. Currently, this seems to be the only way to create a file if it does not exist and otherwise do nothing.

```yaml
---
- name: Create Acme
  ansible.builtin.file:
    path: /srv/traefik/acme.json
    owner: root
    group: root
    state: touch
    mode: '0600'
    access_time: preserve  # to not trigger state changed every time
    modification_time: preserve  # to not trigger state changed every time
```

Listing 7.3. Create Acme Task

7.4.3 MariaDB

One Idea was to roll out the database with a backup automatically. This led to many problems. The first problem encountered was that the file was not readable when copied with the template module. The copy process worked only with the copy module. This gives me the impression that the SQL file had some YAML Syntax, which confused the template-checker.

The read-in process was not difficult to implement, because the database allows for an init folder, in which all files are automatically read-in. The second problem was checking when the database was ready since, after the read-in, the file should be deleted. Usually, the wait-for script would be used for this situation, but this became harder since the database is dockerized.

After trying different ways to check if the read-in process was already finished without success, I pondered one more time the usefulness of the process. If I were to restore the database, I would need to restore all the other services entirely again. Moreover, this would mean that the script can not set up new servers. Bearing this in mind, I simply let the database start and create all the databases and users in the roles of the services, as it will be explained in the next chapter.

7.4.4 Teamspeak

The docker container of Teamspeak uses a user with a unique UID. If the host system does not yet have a user on this UID, the Folders are shown to be owned by the UID. To improve the folder rights appearance, the user module has been used. The module creates the user with the correct UID and creates without a home directory.

Teamspeak needs to be able to write entries into the database. For this purpose, a database and a user need to be created, when the role is called for the first time. At first, a task checks if the database already exists. If so, nothing else needs to be done. Since the database is in docker, it is not possible to rely on a database module for that, and instead, the command module is used, see Figure 7.4. The command module calls into the container where the MySQL client gets all databases. The docker_container_exec module 7.5, was not used here since it led to errors with the MySQL client. From the result, the rows are counted, containing the database name, in this case of teamspeak. To save the result to a variable, the register argument is used.

Sometimes, this task was executed quickly after the database started and did not work. Since Ansible-Roles are separate, a behavior like depends_on in Docker Compose does not exist. The wait-for-module, does not work with docker either which pointed to the need of a solution. The problem is solved by trying six times with a delay of 5 seconds until it does not fail. Yet at first even this always failed.

A problem with this approach was that the task counted as failed when the command block gave back a zero as result, since this is counted the same way as no callback. Because of this a rewrite of the failed_when argument was needed. The new failed_when now checks if the stderr has lines. This fixed the issue.

Lastly, the linter complained that this was a command call, without a defined changed_when. Because of this, it was changed to notify a changed state when the database was not found.

```
- name: Check if DB exists
  command: docker exec mariadb bash -l -c 'mysql -p{{ pw | quote }} -e "SHOW
      ↪ DATABASES;" | grep -c {{ ts_mariadb_database | string }}'
  register: result
  changed_when: result.stdout == 0
  retries: 6
  delay: 5
  failed_when: result.stderr_lines | length > 0  # Try for 30 secs
  until: result is not failed

- name: Create Database
  command: docker exec mariadb bash -l -c 'mysql [...];'
  when: result.stdout == "0"
```

Listing 7.4. Part of the Teamspeak Role

This way, if the result variable is not 0, the "Create Database Task will be skipped.

7.5 Nextcloud

The Nextcloud role contains two handlers. The first installs SVG-support for the Nextcloud instance and needs to run every time the container is rebuilt. As the second handler needs to run only once, I attached it to the folder creation. Both handlers use the docker_container_exec module, allowing running commands inside the container without manual command execution.

The handler 7.5 calls into the Nextcloud container and sets the correct phone region. Since there is a brief moment when the tasks are finished, but the container is not yet reachable, it was necessary to make several attempts and check every time if the command had finished without an error.

```yaml
---
- name: Update Phone Region
  community.docker.docker_container_exec:
    container: nextcloud
    command: "php occ config:system:set default_phone_region --value='DE'"
    chdir: /var/www/html
    user: www-data
  register: result
  retries: 6
  delay: 5
  until: result is not failed
```

Listing 7.5. Change phone region handler

Nextcloud requires a cronjob to make timed tasks, like updating the RSS feeds. The cron module was used, which automatically makes an entry in the user's crontab file. If a name is given to the cronjob, it can be managed by Ansible. Accordingly the interval would change from 5 to 10 minutes and give the same name, Ansible would overwrite the old entry.

7.6 Modularity

To make the script more modular and make the Backup-Role only back up the running services, a system to select the services was introduced. In the all.yml, as seen in Listing 7.6, the user can now change the roles that he/she wants to deploy.

```yaml
---
remote_user: "user"
roles:
      - common
      - mariadb
      [...]
additional_containers:
      - drone-runner-docker
      - drone-runner-ssh

# Backup Storage
retrive_logs: false
backup_path: "/home/user/backup"
remove_after_days: 7
backup_server: url.home
backup_server_port: 22
backup_password: pass
```

Listing 7.6. all.yml

For this, he can edit the roles-variable. The current setup assumes that the names of the roles are the same as the names of the containers. Supposing a container spins up multiple containers; they must be named in the additional_containers variable. In this project, the role that deploys the Drone, the CI/CD, needs two other containers to run docker and ssh pipelines, which are added to the variable.

One notable variable is the remote_user. This would typically be a perfect fit for the server-specific variables. However, those cannot be used for this problem since the remote user needs to be defined in the same step as the host, see Figure 7.7 and the host-specific file is only loaded in during the roles.

```yaml
---
- hosts: netcup
  remote_user: "{{ remote_user }}"
  become: true
  vars:
    ansible_become_pass: "{{ root_pass }}"
  tasks:
    - name: Run Roles
      include_role:
        name: "{{ item }}"
      with_items: "{{ roles }}"
```

Listing 7.7. Full Deployment Play

The typical role argument does not allow looping. Instead, a Task needed to be created, which dynamically loads the roles with the include_role module.

7.7 Backup

The backup is split into two roles. One creates the backup, and another downloads it to a local server. The local server is behind a strict firewall, which does not allow incoming connections, so instead of sending the backup after the creation, the local system downloads the file one hour after the backup job has started.

7.7.1 Backup-Role

The backup role deploys only the backup scripts depending on the currently deployed roles. For this to work, the backup scripts must be called the same as the role.

First, the folders are created, and the base backup script is deployed. Afterwards, a database user with reading rights is created to back up the database if it does not already exist. Then all scripts on the server are checked if the given role is still deployed. This is done by checking whether there are files in the folder that do not match any of the items in the roles-list, see Figure 7.8. In case scripts from old roles are there, they will be deleted.

```yaml
- name: Delete scripts from non-running service
  file:
    path: "{{ item.path }}"
    state: absent
  with_items: "{{ current_files.files }}"
  when: item.path | basename not in backup_names
```

Listing 7.8. Part of the Backup Role

The same is done in reverse for the not-yet-deployed scripts. In the end, the base script is called on a cron job.

The base script, when called, sources all the role-specific scripts to back up the system. All backed-up files are stored in a folder packed to a ZIP-File, which can be downloaded from the backup retrieval role.

7.7.2 Backup-Retrieval-Role

The role first creates its folders, writes a script, which downloads the zip and the log, when specified, and lastly sets up a cron job. For the script creation, the lineinfile module was used, which allows the adding of single lines into a file.

```
- name: Add Backup Log Retrival
  lineinfile:
    path: "{{ backup_path | string }}/retrive.sh"
    line: >-
      scp -o StrictHostKeyChecking=accept-new -P {{ bsp | string }}
      {{ ru | string }}@{{ bs | string }}:/srv/backup/backup.log
      {{ bp | string }}/backups/backup_$(date "+%Y%m%d").log
    state: "{{ 'present' if retrive_logs else 'absent' }}"
```

Listing 7.9. Part of the Backup Retrieval Role

Since this line would be very long and not easy to read, the Block Scalar Operator from YAML was used. The greater-than sign defines that all interior line breaks are filtered out, and the minus removes the line break at the end. [5]

All the variables are defined in the all.yml and are replaced before the file is written on the destination system.

Lastly, the state is set depending on the retrieve_logs variable, defined in the all.yml.

7.8 Stopping and Removing

The play for stopping and removing allows stopping or deleting specific or all the containers. To stop or delete specific containers, the extra vars argument is used, which allows the user to specify extra variables during the ansible-playbook call. The extra vars argument will overwrite any other variable defined in the hierarchy.

The script first gets information from all the current containers on the destination system and saves it in a variable. Afterwards it prefixes the containers either in the roles argument, or the ones specified via the extra variables argument. Since Ansible is built on python, processes like this are made by chaining the commands together.

```
  - name: Create list from containers in the command line
    set_fact:
      container_names: "{{ ([] | zip_longest(containers, fillvalue='/') | map
          ↪ ('join') | list) if containers is defined else
        ([] | zip_longest([container], fillvalue='/') | map('join') | list)
          ↪ }}"
    when: containers is defined or container is defined
```

Listing 7.10. Part of the Removal Play

When the user defines an argument stating that the images should also be deleted, there is no way to know which images are used by the rolled out services and which are not, so the scripts will delete all saved images.

8 Alternatives

There exist alternatives to manage systems and orchestrate Containers, like Puppet or Saltstack.

In comparison, Ansible does not rely on an agent since it organizes itself fully through SSH. That comes with the significant advantage that the server does not need to install software, leading to lower security risk. On the downside, it can not compete with an agent system's speed and scalability advantages. Salt supports a non-client mode, but it currently does not have as many features. [6]

Comparing it to Docker Compose, which is not built to orchestrate whole systems but containers, the next part will discuss Ansible's container management aspects more.

The Docker-Compose file needs to be modified every time a service is enabled or disabled. Since the functions to create something similar to roles are very limited. On the other hand, this simplicity makes it easier for new users to understand the file.

Also, it is not possible to set folder rights or create folder structures before creating the container. This sometimes leads to weird permissions on folders of the server. If a container runs as a user that is not existing outside of the container, the created folder will have a UID that is not bound to a user, like in the case of Teamspeak 7.4.4.

Ansible also allows more freedom by setting up server groups that deploy the script on multiple platforms instantaneously. The remote functions of Docker-Compose are a bit more bare-bones and get mixed with Docker Swarm, the docker alternative to Kubernetes.

Ansible is entirely open-source, while Docker-Compose only open sourced the specifications so far. [7] The Open Source Platform makes it possible for Ansible to build high trust in their security, since it is open to review and thus allows the community to grow faster.

9 Conclusion

During the work, I found that Ansible has many advantages, such as the possibility to use it without installing an agent on the server and running playbooks on multiple server groups without any effort. Another benefit is that Ansible is Open Source, which allows the community to hunt bugs faster and develop deeply integrated features. Another positive aspect is the availability of options while writing playbooks. For example, it is possible to save the variables in a local file from the role and read them from there. Or keep them in a file all the group computers can access. Another example is the possibility of doing everything in one file if the problems are minimal. If it gets bigger later, it is easy to split the one file into a playbook with roles to run. Another positive aspect is that secrets do not need to lie on the server. It is possible to have them locally on the PC and encrypted, and they get only readable during an Ansible deployment. Lastly, it is beneficial that compared to Docker-Compose, it is possible to set up files or folders before deploying the container. That is easy to overlook when setting up new systems and then takes valuable time to fix.

Nonetheless, it also has several pitfalls. The speed of rolling out to many servers is not as high as with agent-based software, leading to problems for enterprise users. Another problem was the bad or deprecated documentation for some topics. Given that the community maintains parts of the documentation, the level of writing varies widely. There have been many deprecation warnings during my works and one even directly after the install 2.2, which is not supposed to occur. Also I would like to mention there are many quirks. An example is the file creation in 7.4.2 or the database problems 7.4.3.

In general, the effort to learn how to create playbooks turns out to be pretty high, so managing a single server may not be worth it. But once the playbooks are written, rolling out features or updating and restarting is just one command line call. Since I now already made the initial time investment, I will stick with Ansible since it speeds up the time to manage the server a good bit.

The future of Ansible may hold more to come. With the open-source approach, it may be possible to click playbooks together with special modules made by the community and active development on the side of Red Hat.

Sources

1. Matt Davis, “Matt on ... whatever: Why no ansible controller for windows?” 2020. [Online]. Available: http://blog.rolpdog.com/2020/03/why-no-ansible-controller-for-windows.html [Accessed: 15.10.2021]
2. R. H. Ansible, “User guide — ansible documentation,” 04.10.2021. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/index.html#getting-started [Accessed: 15.10.2021]
3. Rick Elrod, “Ansible is printing a deprecation warning although target host with ubuntu 20.04 is using python3 · issue #70300 · ansible/ansible,” 2020. [Online]. Available: https://github.com/ansible/ansible/issues/70300#issuecomment-653274921 [Accessed: 15.10.2021]
4. IrinaS, “Ansible: changed=0,” 2019. [Online]. Available: https://stackoverflow.com/questions/54123693/ansible-changed-0 [Accessed: 22.10.2021]
5. jjkparker, “How do i break a string in yaml over multiple lines?” 2010. [Online]. Available: https://stackoverflow.com/questions/3790454/how-do-i-break-a-string-in-yaml-over-multiple-lines/21699210 [Accessed: 3/8/2022]
6. B. Mayer, “Configuration management tools: Ansible, chef, puppet und saltstack im vergleich - golem.de,” *Golem.de*, 09.09.2020. [Online]. Available: https://www.golem.de/news/configuration-management-tools-ansible-chef-puppet-und-saltstack-im-vergleich-2009-150394.html [Accessed: 18.01.2022]
7. “Docker open sources compose specification to accelerate cloud-native application development,” 3/5/2022. [Online]. Available: https://www.docker.com/press-release/docker-open-sources-compose-specification [Accessed: 3/8/2022]