Ordnung im Königreich

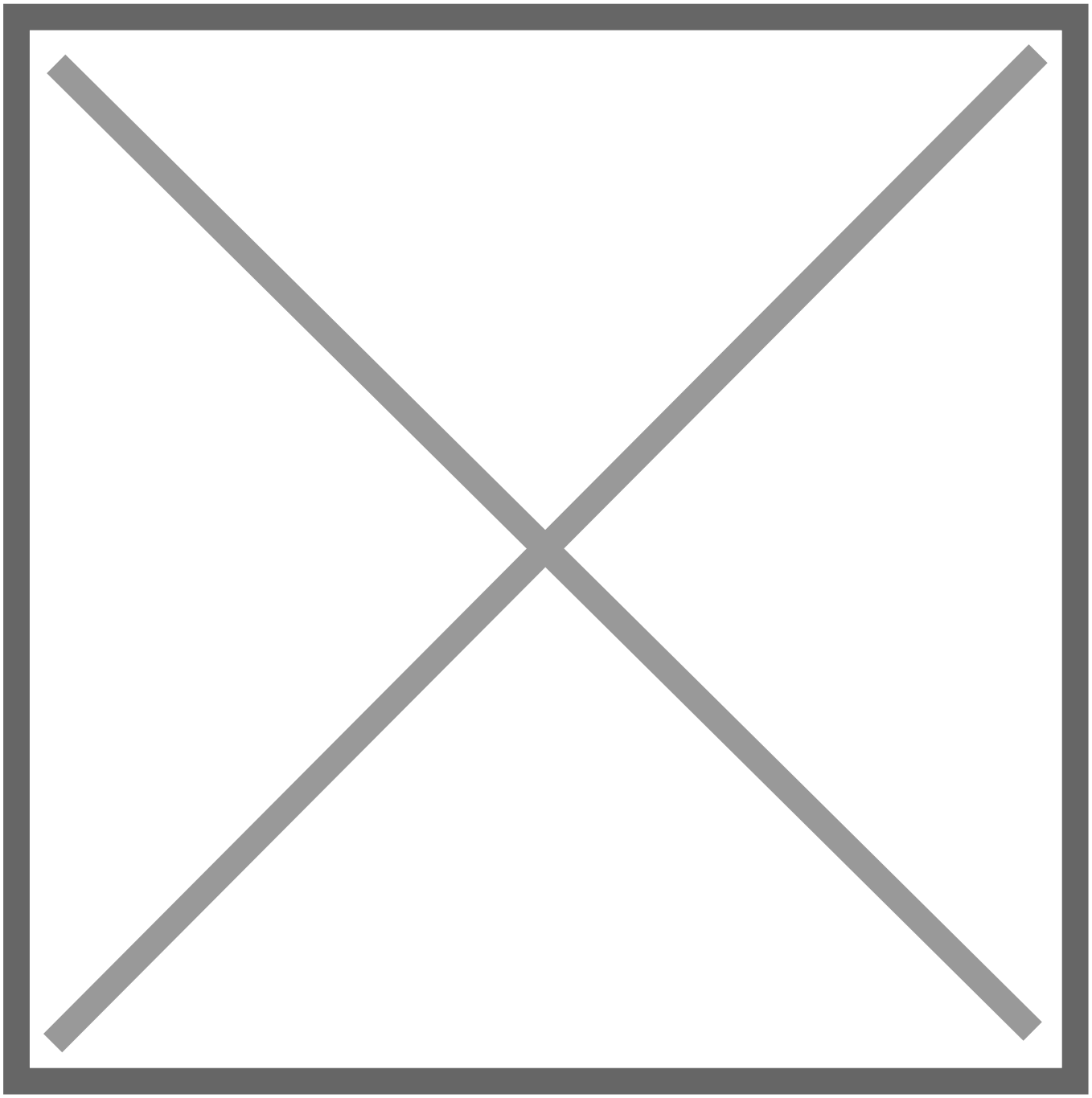


Bild: Thorsten Hübner

Die Grundausstattung für öffentliche Server – automatisiert mit Ansible

Mit einem eigenen Server gewinnen Sie die Datenhoheit zurück. Bevor Sie praktische Dienste installieren können, brauchen Sie aber Zertifikate, einen Reverse-Proxy, automatische Updates und ein Admin-Interface. Mit unserem Server-Baukasten „Telerec’t“ setzen Sie das alles weitgehend automatisch mithilfe von Ansible auf.

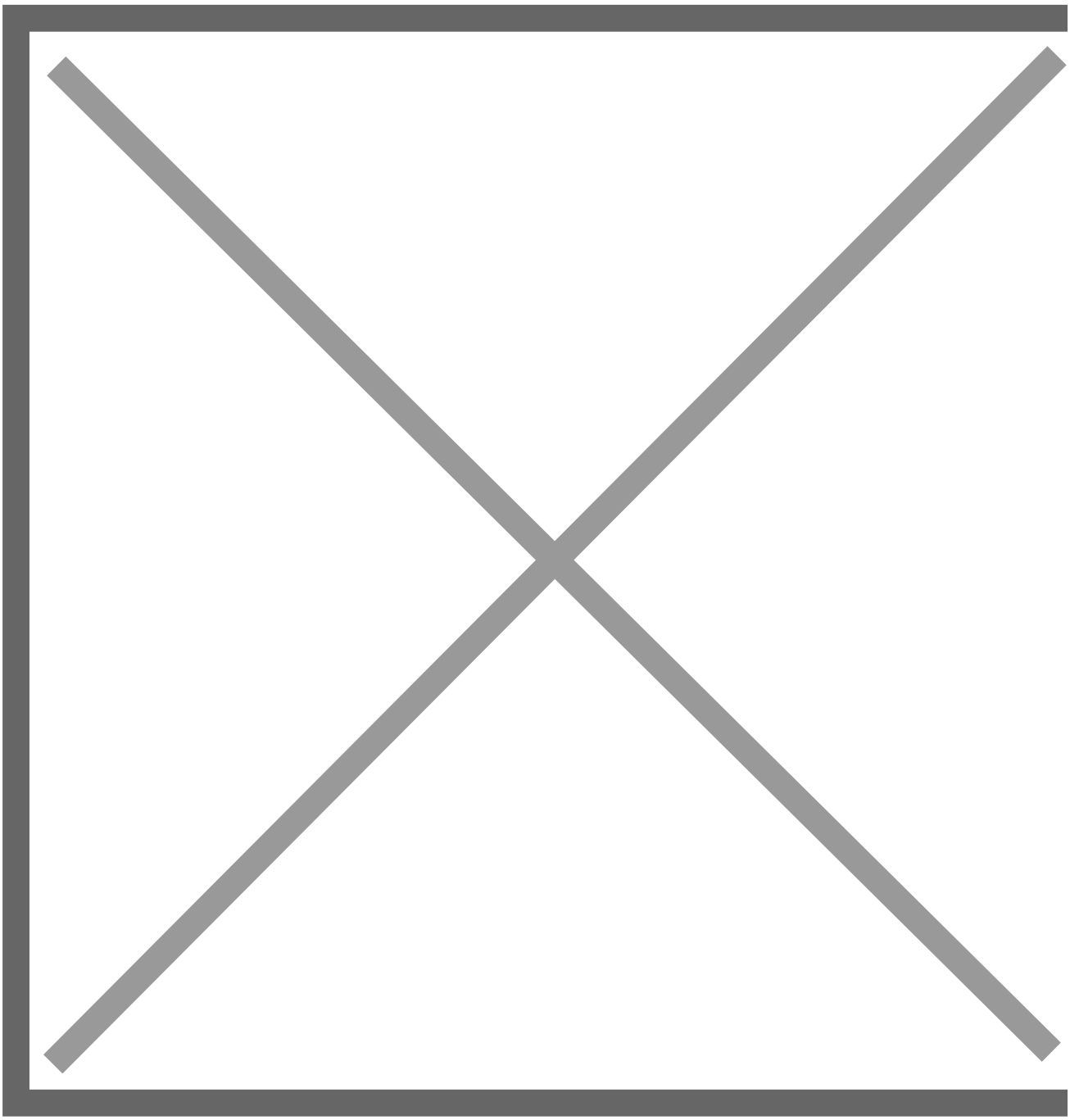
Von Klaus Greff und Pina Merkert
c't 2/2024 Seite 154

c't kompakt

- Ansible richtet vollautomatisch Server ein. Nach der Installation in [1] ergänzen wir in diesem Artikel die Dienste, die jeder im Netz erreichbare Server braucht.
- Heim- und Root- und virtuelle private Server sollten mit signierten Zertifikaten verschlüsselt kommunizieren können, stets aktuell bleiben und leicht administrierbar sein.
- Unser Baukastensystem Telere’ct besteht mindestens aus Docker-Containern für Traefik, Watchtower, Autoheal und Portainer.

Einen Befehl auf der Kommandozeile ausführen, zurücklehnen und zuschauen, wie die eigenen Wünsche umgesetzt werden. Wer seinen Server mit Ansible administriert, kann sich wie ein König fühlen. Wie Sie die ersten Schlachten für die Eroberung Ihres Admin-Reichs schlagen, haben wir in [1] erklärt. Nun gilt es den Server gegen Angriffe abzusichern und einige Helfer für die Verwaltung einzusetzen. Als Monarch lassen Sie nämlich arbeiten und legen nur fest, was Ihre Untertanen zu tun haben.

Den Weg zur Macht beschreiten Sie auf Ihrem Server mit „Teile und Herrsche“. Die Technik dazu heißt „Container“ [2]. Ein Container ist eine auf das Wesentliche zusammengestutzte Linux-Umgebung, die nur einen Dienst ausführt, den aber mit größter Verlässlichkeit. Docker kümmert sich um diese Mini-Linuxe und vernetzt sie virtuell zu einem föderalen Staat. Das ist deutlich effizienter als Virtualisierung, weil alle Container den Kernel des Host mitbenutzen. Der größte Vorteil liegt aber darin, dass Ihr Reich auf den Schultern von Giganten steht: Andere Herrscher teilen bereits optimierte Container-Images über Container-Registries wie den Docker Hub und Sie profitieren von deren Updates.



Regierungsbildung

Damit Ihre Container keinen Staub ansetzen, sollte Ihr Ansible-Reich „Watchtower“ benutzen, das selbst im Container läuft. Es hält Ausschau nach veralteten Docker-Images und ersetzt sie automatisch durch neuere. Dazu gesellt sich „Autoheal“, das regelmäßig prüft, ob alle Container ihre Arbeit verrichten. Wenn nicht, hilft Autoheal ihnen wieder auf die Beine.

Als Außenminister empfehlen wir den Reverse-Proxy „Træfik“ (im Folgenden Traefik geschrieben), der verlässlich alle Anfragen befreundeter Rechner annimmt und an die Container weiterleitet, die dafür zuständig sind. Traefik regelt auch die Transportverschlüsselung der Kommunikation, indem es SSL-Zertifikate von Let’s Encrypt beschafft und automatisch vor Ablauf verlängert.

Vielleicht befürchten Sie, dass es unübersichtlich werden könnte, die vielen Container gleichzeitig im Auge zu behalten. Zum Glück schafft „Portainer“ mit seinem Dashboard Abhilfe. Die Webanwendung listet all Ihre Container auf, erleichtert deren Konfiguration und hilft mit Logs bei der Fehlersuche. Also keine Sorge, dass Sie zu diesem Zweck Ihr Reich zu Fuß bereisen oder mit Konsolenbefehlen hantieren müssten.

Das Repository als Verfassung

Ansible steht Ihnen stets als Protokollchef zur Seite, der den Kleinkram für Sie verwaltet. Wir gehen im Folgenden davon aus, dass Sie Ansible installiert und ein Repository für Ihr Setup angelegt haben. Falls Sie das schnell nachholen wollen, finden Sie alle Infos in [1]. Ein Ansible-Setup besteht immer aus drei Bereichen: Variablen, Roles und Playbooks.

Mit den Variablen legen Sie all die Werte fest, die für Ihr Setup individuell sind. Das sind Benutzernamen, Passwörter, Pfade und Abweichungen von der Standardkonfiguration. Falls eine Variable nur einen bestimmten Server betrifft, legen Sie diese im Ordner host_vars/ in einer YAML-Datei fest, deren Name dem Hostnamen oder der IP-Adresse des Servers entspricht. In unserem in [1] beschriebenen Basis-Setup haben wir so unter anderem den Namen des Standardbenutzers und die Zeitzone des Servers in host_vars/192.168.7.140.yml festgelegt.

> In group_vars/all.yml erstellen Sie einen Block für jeden Dienst, der auch Geheimnisse wie Pas
> In group_vars/all.yml erstellen Sie einen Block für jeden Dienst, der auch Geheimnisse wie Passwörter enthalten darf. Die compose_hull lädt diese Variablen und verbindet Sie mit Standardwerten der Role. Für das Basis-Setup tragen Sie in diese Datei Ihre neu erzeugten Passwörter ein.

Gelten Einstellungen für mehrere Server, sind sie in group_vars/all.yml besser aufgehoben. Dort haben wir Abschnitte für jeden unserer Serverdienste erstellt und dort beispielsweise alle Benutzernamen und Passwörter festgelegt. Da unser Basis-Repository privat ist, können wir damit Passwörter über mehrere Rechner, von denen wir den Server administrieren, synchron halten, beispielsweise der Desktop-PC im Büro und das private Notebook. Falls Ihnen das zu unsicher ist, müssen Sie die Passwortverwaltung in eigene Dienste auslagern. Damit dieser Artikel nicht zu lang wird, haben wir uns gegen diesen Weg entschieden und riskiert, dass GitHub unsere Passwörter kennt.

In unserem Vorlagen-Repository github.com/ct-Open-Source/telerec-t-base enthält die Datei schon Blöcke mit den Variablen der vier Container, die Sie immer brauchen. Die Passwörter müssen Sie aber unbedingt ändern! Der folgende Linux-Befehl erzeugt ein Passwort mit 25 Zeichen und nutzt dafür die besten Zufallszahlen, die der Kernel erzeugen kann:

```
cat /dev/urandom | tr -dc a-zA-Z0-9 | head -c25; echo
```

Nutzen Sie diesen Befehl, um ein `admin_password` für `portainer`, ein `http_token` für `watchtower` und ein `admin`-Passwort für `traefik` zu erzeugen. Letzteres müssen Sie für die HTTP-Basic-Authentifizierung des Webinterfaces von Traefik noch hashen und mit dem Nutzernamen

kombinieren. Der folgende Befehl macht das beispielhaft für den Benutzernamen `admin` und das Passwort `Geheimnis`:

```
htpasswd -nb admin Geheimnis | sed -e s/\\$/\\$\\$/g
```

Die Ausgabe dieses Befehls fügen Sie als Variable `http_basic_users` im Block `traefik` ein.

Sie können auch Ihren Passwortmanager benutzen, um sichere Passwörter für den Server zu erzeugen. Unter Windows und macOS würden wir von vornherein einen Passwortmanager wie KeePass [2] oder Bitwarden [3] zum Erzeugen der Passwörter empfehlen (beide laufen ebenfalls unter Linux). Da Sie die meisten Passwörter nie eingeben müssen, können Sie die extrem lang erzeugen lassen. Ausnahme: Das Passwort für BasicAuth brauchen Sie für jeden Login am Traefik-Dashboard.

Mit den Roles konfigurieren Sie, welche Dienste zu Ihrem Setup gehören. Dazu können Sie sich an unseren Vorlagen bedienen und Sub-Repositories in den Ordner roles/ klonen. Das geht mit `git submodule add` – ein vollständiges Beispiel hatten wir in [1] angegeben. Alternativ können Sie auch ohne Sub-Repository einfach Ordner in roles/ anlegen und selbst neue Dienste konfigurieren.

Die letzte Zutat sind Playbooks, die Sie im Wurzelverzeichnis Ihres Repository anlegen. Sie sind so kurz, dass es sich nicht lohnt, sie gesondert zu synchronisieren. Hier ein Beispiel für ein Playbook, das die Role für Traefik ausführt:

```
- hosts: server
  become: true
  roles:
    - role: traefik
      vars:
        service_cfg: "{{ traefik }}"
```

Die ersten drei Zeilen sind immer gleich. Danach kommt die YAML-Liste aller Roles, die das Playbook ausführt. Die Angabe hinter `vars:` setzt die Variablen aus dem `traefik`-Block definiert in group_vars/all.yml. Die Zeile weist der Variable `service_cfg` den Inhalt des Blocks zu und die Compose Hull ergänzt danach diesen Block. Wir nutzen solche nahezu identischen Playbooks mit genau einer Role, um Dienste bei Bedarf einzeln neu starten zu können, beispielsweise nachdem wir eine Variable geändert haben. Ein Playbook, das gleich mehrere Dienste aufsetzt, sähe genauso aus mit einer Liste mit mehr Spiegelstrichen hinter `roles:`.

„Aufsetzen“ und „Neustarten“ nutzen wir bei Ansible übrigens synonym. Ansible-Playbooks dienen nämlich immer dazu, einen bestimmten Zustand auf dem Server herzustellen. War ein Dienst dort nicht installiert, müssen die im Playbook definierten Roles alle Dateien und Ordner anlegen und den Dienst starten. Lief der Dienst bereits mit einer minimalen Änderung an der Konfiguration, überprüft Ansible nur, dass die Ordner und Dateien existieren und startet mit der geänderten Konfiguration neu. Wenn Sie selbst Roles schreiben, müssen Sie daran denken, dass die auch genauso funktionieren und nicht jeder zusätzliche Durchlauf einen bereits korrekten Zustand auf dem Server noch mal verändert.

Compose Hull

Der raffinierte Teil von Telerec't steckt in der Role `compose_hull`, die selbst nie in einem Playbook steht. Die Motivation: Wenn Sie im Netz auf die Suche nach attraktiven Serverdiensten gehen, werden Sie inzwischen fast immer auch Docker-Container und ein passendes docker-compose.yml finden. Wir wollten es so einfach wie möglich machen, Dienste mit diesen Infos auch mit Ansible zum Laufen zu bringen. Das Vorgehen dafür sieht folgendermaßen aus: Sie kopieren die Datei docker-compose.yml in eine leere Role (optional können Sie sie in den Unterordner templates/ stecken) und benennen sie um in docker-compose.yml.j2. Ab sofort ist sie ein Jinja2-Template, in dem Sie in doppelten geschweiften Klammern alle Ansible-Variablen benutzen können. So brauchen Sie kein Copy & Paste.

Gehen Sie dann die Datei durch und lagern Sie Geheimnisse wie Passwörter nach group_vars/all.yml in einen neuen Block mit dem Namen des Diensts aus. Kommen Werte in docker-compose.yml vor, die nicht geheim sind, erstellen Sie für diese Variablen in defaults/main.yml unterhalb von `service-defaults:` (siehe unten). So vermeiden Sie Fehler beim Copy & Paste, wenn sie die Werte mal anpassen müssen. Eine Variable `directory` referenzieren Sie im Template beispielsweise mit `{{ service_cfg.directory }}`.

Damit die Dienste vom Reverse-Proxy Traefik verschlüsselt und mit Zertifikaten ausgestattet werden, muss man eine ganze Reihe von Labels definieren und den Container, der von außen erreichbar ist, in das Docker-Netzwerk von Traefik einbuchen. Mit der Compose Hull geht das mit zwei simplen YAML-Referenzen auf vordefinierte Blöcke:

```
  labels: *base_labels
  networks: *base_networks
```

Falls Sie weitere Labels hinzufügen wollen, können Sie die YAML-Syntax mit `<<` verwenden:

```
  labels:
    << : *base_labels
        {# weitere.labels: "hier" #}
```

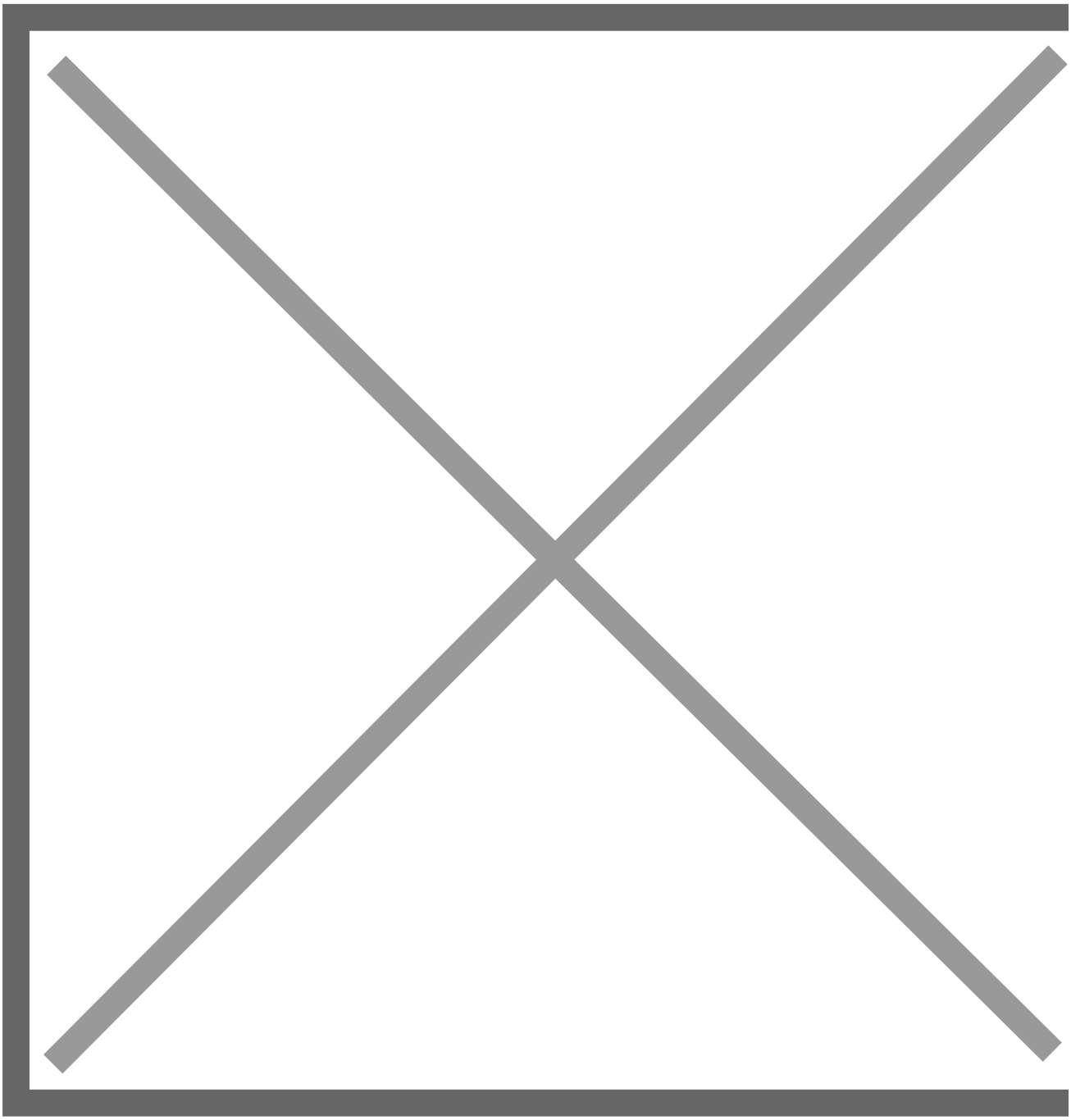
Um die Magie der Compose Hull zu nutzen, fügen Sie einen Task wie diesen in Ihre Role ein:

```
- ansible.builtin.import_role:
    name: compose_hull
  vars:
    service_defaults:
      directory: "{{ docker_dir }}/2fauth"
      name: twofauth
      traefik: true
      external: true
      watchtower: true
      autoheal: true
      skip_network_definition: false
```

Das Playbook dazu sieht dann wie folgt aus:

```
- hosts: server
  become: true
  roles:
    - role: twofauth
      vars:
        service_cfg: "{{twofauth_cfg}}"
```

Beim Aufruf fügt die Compose Hull zunächst alle Variablen in einen Block `service_cfg` zusammen. Entgegen Ansibles Voreinstellung überschreiben hier die Geheimnisse aus group_vars/all.yml die Voreinstellungen in der Role selbst. Einen Block mit passendem Namen (im Beispiel `twofauth_cfg`) muss es geben, er darf aber leer sein. Danach legt die Role falls nötig das `directory` und alle `subdirs` an. Dann baut die Compose Hull eine docker-compose.yml mit den Label- und Netzwerk-Blöcken und ersetzt dabei auch alle Jinja2-Variablen. Zuletzt führt der Task `community.docker.docker_compose` die Container-Konfiguration auf dem Server mit `docker-compose` hoch. Mit den Ansible-Tags `started`, `restarted`, `recreate` und `stopped` können Sie `docker-compose` auch Statusbefehle mitgeben. `started` ist die Standardeinstellung.



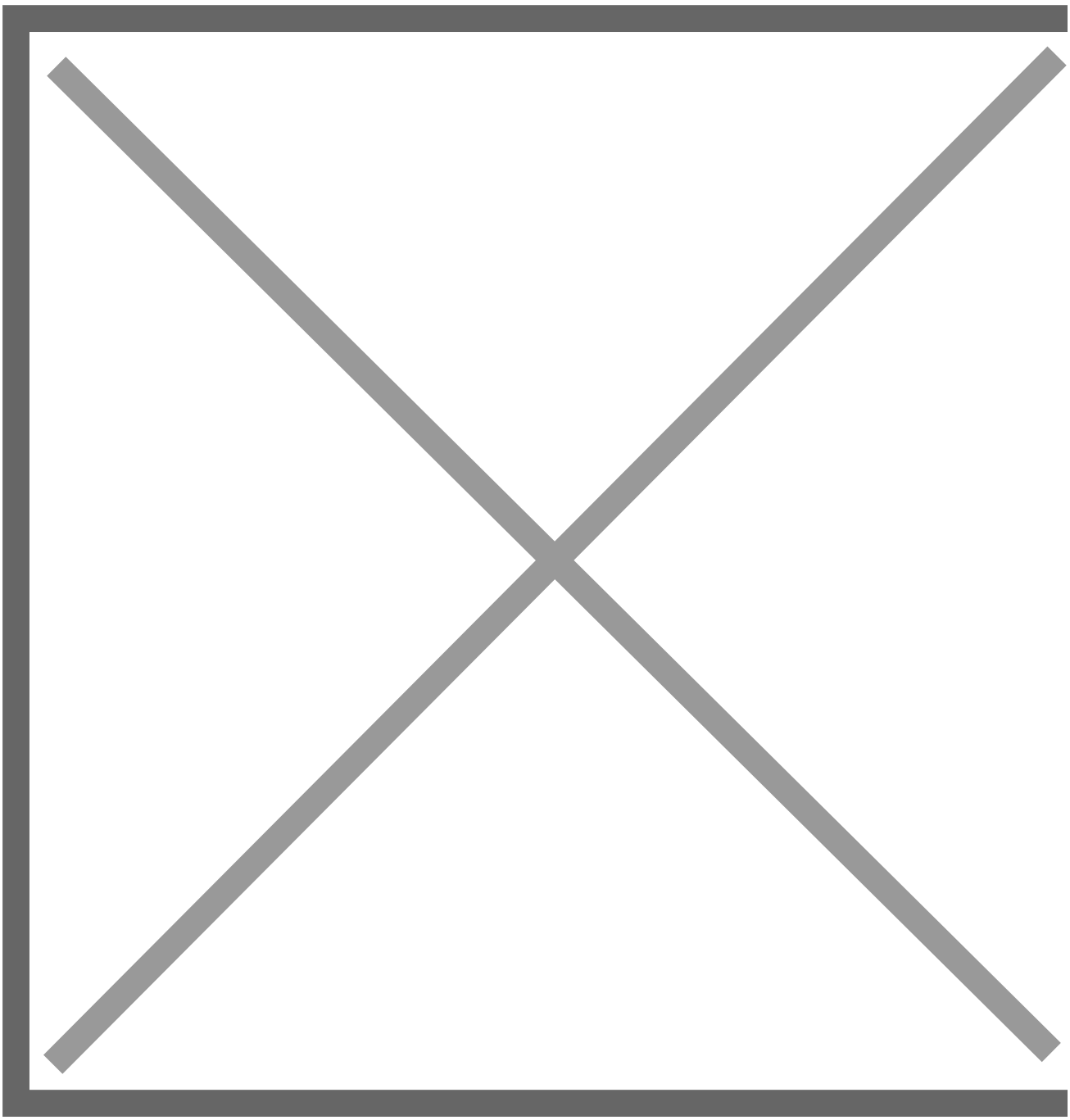
Traefik

Ein Proxy-Server vermittelt Anfragen zwischen Clients und Webservern. Ein Reverse Proxy macht das Gleiche auf der Serverseite. Er schaut bei Anfragen an die gleiche IP-Adresse nach, an welche Subdomain die eigentlich gerichtet waren und leitet die HTTP-Anfragen an den Webserver im passenden Container weiter.

In diesem Betriebsmodus läuft Traefik in Telerec’t. Zusätzlich kann er sich als zentraler Vermittler auch darum kümmern, Zertifikate auszuhandeln und den verschlüsselten Datenverkehr von den Clients schon mal zu entschlüsseln. Die Webserver in den Containern müssen dann keine

Zertifikate verwalten und bei unverschlüsselten Anfragen nicht auf HTTPS weiterleiten. Will man den Zugriff auf einen Webdienst mit HTTP-Basic-Authentifizierung einschränken, kann das Traefik mit einer Middleware namens BasicAuth übernehmen. Sie aktivieren diese, indem Sie einfach ein Label in der Container-Konfiguration ergänzen.

Um neue Dienste zu integrieren, müssen Sie die Konfigurationsdateien von Traefik nicht verändern. Stattdessen reagiert der Reverse Proxy auf bestimmte Labels, die direkt beim Container stehen. Das macht die Konfiguration angenehm modular. Mit einem Webinterface samt Dashboard informiert Traefik über die verwalteten Dienste und die jeweils genutzten Middlewares.

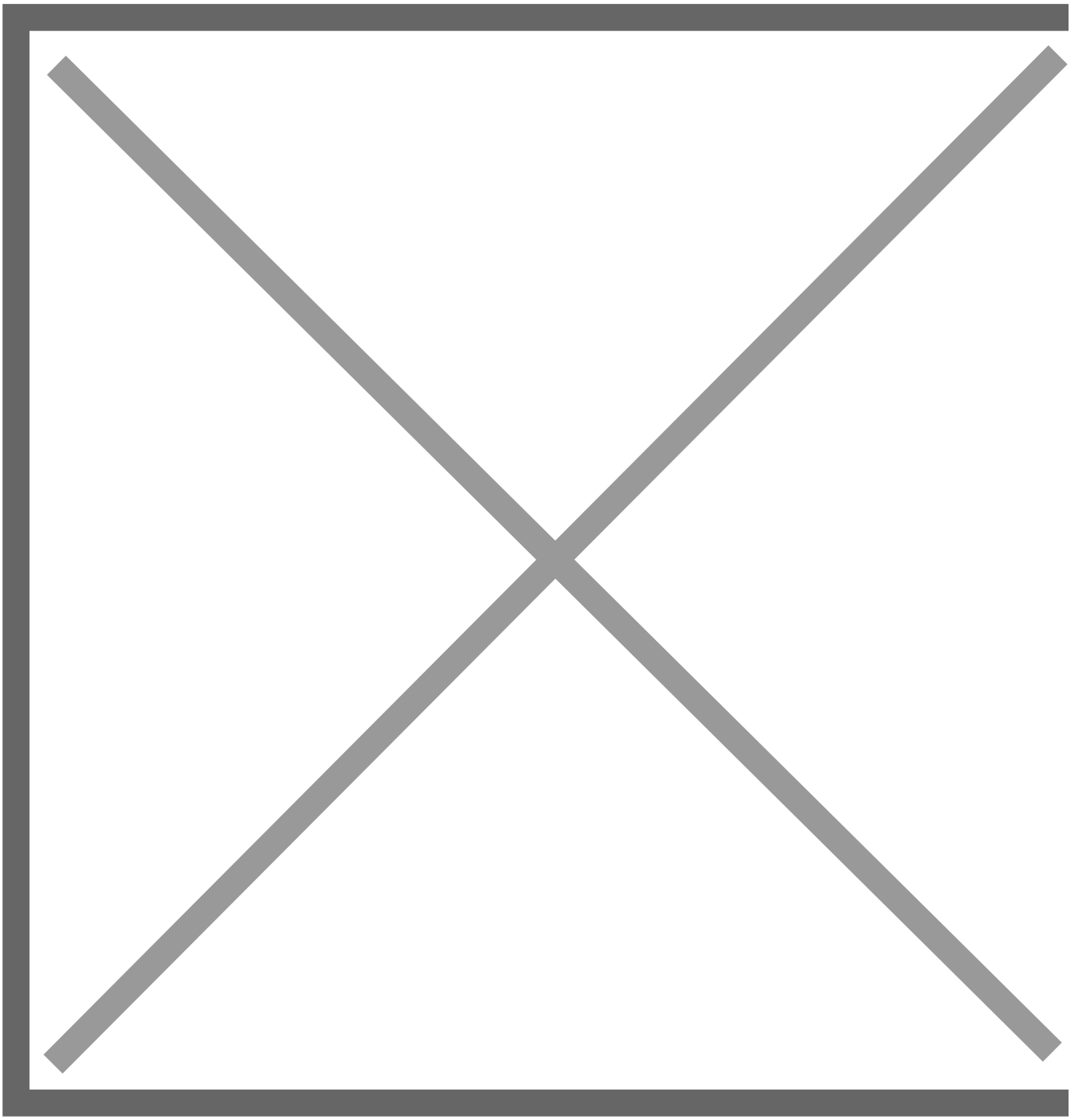


Watchtower und Autoheal

Auch für erfahrene Admins sind Sicherheitslücken höhere Gewalt. Weil es zu lange dauern würde, einen eigenen Patch zu entwickeln, warten Admins auf das nächste Update und spielen es sofort ein. Für große IT-Infrastruktur stehen dafür rund um die Uhr Admins bereit, beim eigenen Server ist die Gefahr aber groß, das Sicherheitsupdate erst verzögert einzuspielen.

Ein typisches Problem für eine weitere Schicht der Automatisierung: Watchtower überwacht die Registries, üblicherweise den Docker Hub, ob neue Images der installierten Container bereitstehen. Wenn ja, lädt die Software sie automatisch und tauscht sie im laufenden Betrieb aus. Dabei hält sich die Software an die im Compose-File definierten Tags, sodass man große Versionssprünge unterbinden kann, wenn man Probleme erwartet. Legen Sie die Tags aber nicht auf eine einzelne Version fest, weil das Sicherheitsupdates verhindern kann und setzen Sie auf Container, deren Maintainer Updates der Dienste auch verlässlich in die Images einbauen.

Autoheal kämpft ergänzend gegen Downtimes, indem es abgestürzte Container neu startet. Ob ein Container läuft, prüft ein hinter `healthcheck:` definierter Befehl. Das kann beispielsweise ein `wget`-Aufruf auf ein Webinterface sein. Gut programmierte automatische Tests können hier auch irreguläre Betriebszustände erkennen. Der Healthcheck prüft nur den Status-Code.



Portainer

Die Macht, neue Container ruckzuck auszuprobieren und mit einem einzigen Konsolenbefehl automatisiert aufzusetzen, wird Sie dazu verleiten, in kurzer Zeit viele Container anzusammeln. Konsolenbefehle wie `docker ps`, der alle laufenden Container auflistet, erfordern nicht nur ein SSH-Login, sie helfen auch kaum bei der Übersicht, wenn sich die Ausgabe über mehrere Bildschirmseiten erstreckt.

Mit Portainer kommt die Übersicht zurück. Die intuitive Weboberfläche informiert Sie, welche Container gerade laufen oder eben nicht. Praktisch ist, dass alle Container, die im gleichen Compose-File definiert wurden, auch in einen Stack gruppiert werden. Portainer zeigt mit wenigen Klicks Einstellungen und Logfiles und öffnet auf Wunsch direkt im Browserfenster eine Shell innerhalb eines Containers. Die Suche nach Problemen geht mit den von Portainer bereitgestellten Werkzeugen wesentlich flotter. Auch verwaiste Netzwerke und Images sind viel schneller aufgeräumt, als mit Dutzenden Konsolenbefehlen.

Portainer greift in Telerec't seine Infos über den Docker-Socket direkt ab, sodass Sie ohne weitere Konfiguration loslegen können. Ein Remote-Zugriff wäre mit der Software auch möglich, Telerec't braucht das aber nicht. Wer nicht alleine administriert, kann mit dem Admin-Account weitere Accounts anlegen. Das zwischen hell und dunkel einstellbare Theme passt sich automatisch der Systemeinstellung an.

Ausführung gefällig?

Unsere Erklärung der Basisdienste kommt ohne Installationsbefehle aus, weil wir alle vier Dienste bereits als Submodules im Basis-Repository definiert haben. Zur Erinnerung: Im ersten Teil des Artikels haben Sie die Submodules mit folgendem Befehl auf den Rechner geholt:

```
git submodule update --init \
  --recursive
```

Traefik-Konfiguration für Heimserver

Falls Sie für den Heimserver in den eigenen vier Wänden interne Dienste mit priviligiertem Zugriff aus dem Heimnetz einrichten möchten, geben Sie in der Liste `traefik.internal_ip_ranges` in group_vars/all.yml die IP-Bereiche an, in denen Ihr Router Adressen vergibt. Dienste mit der Variable `external: false` sind dann von Geräten mit internen IP-Adressen weiterhin erreichbar. Das bietet sich beispielsweise für Smart-Home-Container an, deren Ports Sie nicht im Internet veröffentlichen möchten. Alternativ können Sie diese Dienste auch extern freigeben und eine Authentifizierung wie bei der Weboberfläche von Traefik davor schalten.

Danach gibt es die Unterverzeichnisse in roles/ und Sie können die Roles in Playbooks benutzen. Die Playbooks für die vier Basisdienste haben wir im Basis-Repository schon vorbereitet und Sie können die vier Dienste nacheinander hochfahren:

```
pipenv run ansible-playbook traefik.yml -i hosts
pipenv run ansible-playbook watchtower.yml -i hosts
pipenv run ansible-playbook autoheal.yml -i hosts
pipenv run ansible-playbook portainer.yml -i hosts
```

All das geht alternativ auch in einem einzigen Befehl:

```
pipenv run ansible-playbook server-setup.yml -i hosts --ask-pass --ask-become-pass
```

Ihr Reich hat jetzt eine einwandfreie Infrastruktur. Sie dient allerdings bislang nur dem Zweck, sich selbst zu betreiben. Mit dieser Basis können Sie nun jedoch mit gutem Gewissen die Dienste installieren, für die Sie den Server eigentlich haben wollten. Nützlich finden wir beispielsweise: Nextcloud für Dateisynchronisation, Wordpress für einen privaten Blog, Vaultwarden für die Passwortverwaltung, NodeRed und Mosquitto für das Smart Home oder Wekan für private Kanban-Boards. Wenn wir in den folgenden Ausgaben solche Dienste mit Ansible aufsetzen, verweisen wir immer auf diesen Artikel, weil die Basis immer gleich ist, egal für welche dieser Dienste Sie sich entscheiden.

Unsere Hoffnung ist, dass unser Beispiel-Setup Telerec't Ihnen nicht nur Arbeit bei der Server-Administration abnimmt, sondern Sie auch Lust bekommen, das Setup mit eigenen Roles als Submodule zu erweitern. Von Ihren öffentlichen Repositories können dann auch andere profitieren und das Baukastensystem wächst und gedeiht. Wir verlinken Ihre Submodule-Repositories gern in der Readme-Datei unseres Basis-Repository. Schreiben Sie uns einfach eine kurze Nachricht mit dem Link an pmk@ct.de. *(pmk@ct.de)*

1. Literatur

2. Klaus Greff und Pina Merkert, Telerec't, Ein eigener Server im Rechenzentrum oder daheim – automatisiert mit Ansible, c't 1/2024, S. 150
3. Marvin Strathmann, Schlüsselfertig, So bringen Sie Ordnung ins Passwort-Chaos, c't 15/2019, S. 172
4. Niklas Dierking, Geheimniskrämer, Der Raspberry Pi als Passwort-Server, c't 9/2021, S. 18
5. Jan Mahn, Container-Komponist, Docker-Container mit Docker-Compose einrichten, c't 6/2018, S. 148
6. Jan Mahn, Container-Bedienpulte, Grafische Oberflächen für Docker, c't 6/2019, S. 158
7. Jan Mahn, HTTP-Einweiser, Eingehenden HTTP-Verkehr mit Traefik routen, c't 17/2019, S. 158
8. Merlin Schumacher, Container cum laude, Empfehlenswerte und gut gepflegte Docker-Container für den Alltag und als Inspiration, c't 16/2018, S. 108
9. Jan Mahn und Peter Siering, Container mit Docker und Docker-Compose, c't 14/2022, S. 186
10. Markus Stubbig, Cloudtresor zum Generieren von Einmalpasswörtern, 2FA-Authentifikator selbst gemacht, c't 27/2023, S. 146

Repositories: ct.de/yruq

Revision #1
Created 2026-09-26 13:20:54 UTC by ralf
Updated 2026-09-26 13:20:54 UTC by ralf