HAW HAMBURG

BACHELORTHESIS
Bastian Schlüter

Evaluation von Software-Lösungen zur automatisierten Server-Administration

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Bastian Schlüter

Evaluation von Software-Lösungen zur automatisierten Server-Administration

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Jens von Pilgrim
Zweitgutachter: Prof. Dr. Bettina Buth

Eingereicht am: 23. Ferbruar 2022

Bastian Schlüter

Thema der Arbeit

Evaluation von Software-Lösungen zur automatisierten Server-Administration

Stichworte

Ansible, Chef, Automatisierung, Push, Pull, Playbook, Play, Cookbook, Recipe

Kurzzusammenfassung

Zeit in die Automatisierung häufiger Arbeitsabläufe zu investieren, kann sich schnell auszahlen. Im Folgenden werden deshalb *Ansible* und *Chef* anhand realitätsnaher Fallbeispiele gegeneinander abgewogen, um eine fundierte Entscheidungsgrundlage für die Auswahl einer Automatisierungs-Lösung zu liefern. Dabei zeigt sich, dass *Ansible* durch seine intuitive Web-Oberfläche und die Nutzung eines Push-Ansatzes, besonders für Anfänger, die bevorzugte Wahl sein sollte.

Bastian Schlüter

Title of Thesis

Evaluation of Software Solutions for automated Server Administration

Keywords

Ansible, Chef, Automation, Push, Pull, Playbook, Play, Cookbook, Recipe

Abstract

Investing time in automating common workflows can quickly pay off. In the following, *Ansible* and *Chef* are therefore weighed against each other using realistic case studies in order to provide a well-founded basis for selecting an automation solution. It turns out that *Ansible* should be the preferred choice, especially for beginners, due to its intuitive web interface and the use of a push approach.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

Abkürzungen **viii**

Quellcodeverzeichnis **ix**

1 Einleitung **1**

1.1 Problemstellung 1
1.2 Zielsetzung 1
1.3 Auswahlkriterien 2
1.4 Bewertungskriterien 2
1.4.1 Functional Suitability 2
1.4.2 Compatibility 3
1.4.3 Usability 3
1.4.4 Security 3
1.4.5 Maintainability 3
1.4.6 Portability 3

2 Betrachtete Software-Lösungen **4**

2.1 Ansible 4
2.1.1 Allgemein 4
2.1.2 Begriffsbildung 4
2.1.3 Verteilung 4
2.1.4 Installation 6
2.1.5 Sprache 8
2.1.6 Grundlegende Funktionsweise 8
2.1.7 Konzeptioneller Aufbau 9
2.2 Chef 13
2.2.1 Allgemein 13

2.2.2 Begriffsbildung . . . 13
2.2.3 Verteilung . . . 14
2.2.4 Installation . . . 14
2.2.5 Sprache . . . 15
2.2.6 Grundlegende Funktionsweise . . . 15
2.2.7 Konzeptioneller Aufbau . . . 15

3 Fallstudien **19**
3.1 Einrichten eines Web-Servers . . . 19
3.1.1 Ansible . . . 20
3.1.2 Chef . . . 20
3.1.3 Auswertung . . . 21
3.2 Automatisches Erneuern von SSL-Zertifikaten . . . 21
3.2.1 Ansible . . . 22
3.2.2 Chef . . . 25
3.2.3 Auswertung . . . 26
3.3 Sequenzielles Herunterfahren von Diensten auf verschiedenen Systemen . . . 27
3.3.1 Ansible . . . 28
3.3.2 Auswertung . . . 28

4 Diskussion **29**
4.1 Auswertung . . . 29
4.1.1 Functional Suitability . . . 29
4.1.2 Compatibility . . . 29
4.1.3 Usability . . . 30
4.1.4 Security . . . 32
4.1.5 Maintainability . . . 33
4.1.6 Portability . . . 33
4.2 Probleme . . . 34
4.2.1 Ansible . . . 34
4.2.2 Chef . . . 35

5 Fazit **36**
5.1 Bewertung . . . 36
5.1.1 Ansible . . . 36
5.1.2 Chef . . . 36
5.2 Empfehlung . . . 37

5.3 Grenzen der Automatisierung 37

Literaturverzeichnis **38**

Glossar **41**

Selbstständigkeitserklärung **44**

Abbildungsverzeichnis

1.1 ISO 25010 - Qualitäts-Kriterien für Software [10] 2

2.1 Ansible - AWX Verteilungs-Diagramm 5
2.2 Ansible - Ausführung eines Modules 8
2.3 Ansible - AWX Klassen-Diagramm 9
2.4 Ansible - Example Playbook Aktivitäts-Diagramm 12
2.5 Chef - Verteilungs-Diagramm 14
2.6 Chef - Ausführen einer Run-List 15
2.7 Chef - Klassen-Diagramm 16
2.8 Chef - Example Recipe Aktivitäts-Diagramm 18

3.1 Install Apache Aktivitäts-Diagramm 19
3.2 ACME-Abstraktion Aktivitäts-Diagramm 21
3.3 Ansible - Renew Certificate Aktivitäts-Diagramm 24
3.4 Chef - Renew Certificate Aktivitäts-Diagramm 26
3.5 Shutdown Services Aktivitäts-Diagramm 27

Abkürzungen

SSH Secure Shell.

ACME Automatic Certificate Management Environment.

HTTP Hyper Text Transfer Protocol.

SSL Secure Sockets Layer.

Quellcodeverzeichnis

1 Ansible - AWX - Kubernetes-Cluster auf lokaler Maschine bereitstellen . . 6
2 Ansible - Ausrollen des AWX Operators 7
3 Ansible - examplePlaybook.yml 12
4 Chef - Installation von *Chef Automate* und *Chef Infra* 14
5 Chef - exampleRecipe.rb 17

6 Ansible - installApache.yml 20
7 Chef - installApache.rb 20
8 Ansible - renewCertificate.yml 22
9 Ansible - installApache.yml Fortsetzung 23
10 Chef - renewCertificate.rb 25
11 Ansible - shutdownServices.yml 28

1 Einleitung

1.1 Problemstellung

Die IT-Welt ist in stetigem Wandel. Anforderungen steigen und mit ihnen auch die Komplexität von Server-Landschaften. Ausfall-Sicherheit und Load-Balancing sind Gründe, für das Ausrollen von Diensten mehrere Server in Betracht zu ziehen. Doch mit Anzahl und Diversität steigt auch der zeitliche Aufwand für die Verwaltung und Wartung. Gegebenenfalls ist es möglich, die gestellten Anforderungen auch per eigenem Script zu erfüllen. Jedoch bieten zentralisierte Software-Lösungen eindeutige Vorteile bezüglich Multi-Plattform, Wartbarkeit, Reporting, Sicherheit und der Zusammenarbeit als Team.

1.2 Zielsetzung

Diese Arbeit betrachtet Kosten und Wert der Nutzung vorhandener Produkte zur automatisierten Server-Administration, wägt diese gegeneinander ab und bietet Empfehlungen bezüglich optimaler Use-Cases der einzelnen Lösungen. Themen wie Sicherheit, Bedienungsfreundlichkeit, sowie genutzte Technologien und deren Relevanz in der heutigen IT-Welt werden ebenfalls berücksichtigt. Weiterhin werden Grenzen der (sinnvollen) Automatisierung aufgezeigt und durch betriebsnahe, reale Beispiele gestützt.

Weiterhin soll diese Arbeit als Hilfestellung dienen, um System-Administratoren einen Überblick über die vorgestellten Software-Lösungen zu gewähren und ihnen zu helfen, bei der Auswahl der vorgestellten Tools eine informierte Entscheidung zu treffen. Ziel dieser Arbeit ist es jedoch nicht, eine lückenlose Schritt für Schritt Anleitung zu sein. Für diesen Zweck sind weiterhin die entsprechenden Dokumentationen aufzusuchen.

1.3 Auswahlkriterien

Die Auswahl der betrachteten Software-Lösungen geschieht unter dem Bestreben, ein möglichst großes Feature-Set abzudecken und somit einen Überblick über mögliche Herangehensweisen der Server-Administration zu liefern.

So unterscheiden sich *Ansible* und *Chef* beispielsweise grundlegend in ihrem Kommunikations-Protokoll. Während *Ansible* einen *Push*-Ansatz bei der Kommunikation mit den zu steuernden System verfolgt, setzt *Chef* hingegen auf einen *Pull*-Ansatz [28].

Weiterhin bietet *Ansible* mit *AWX* die Möglichkeit, zu einem Großteil über ein Web-Interface verwaltet zu werden. *Chef* hat durch *Chef Automate* ein Dashboard, welches beinahe ausschließlich zur Beobachtung des Systems dient.

1.4 Bewertungskriterien

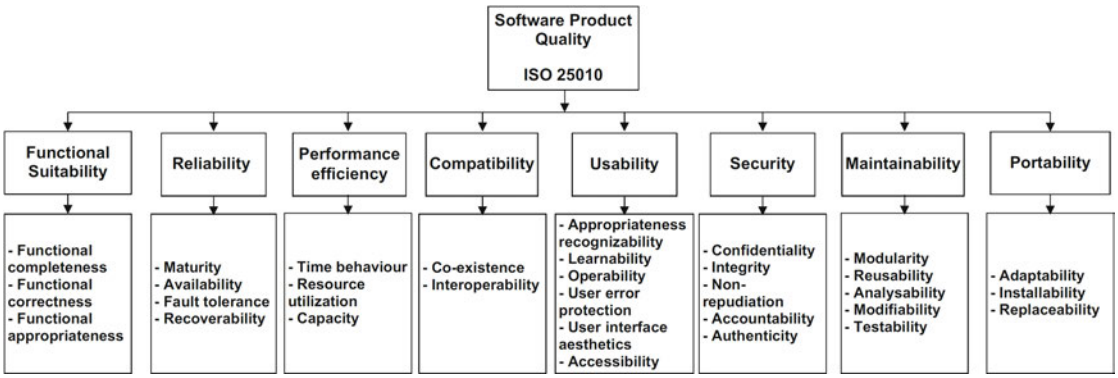

Abbildung 1.1: ISO 25010 - Qualitäts-Kriterien für Software [10]

Die Bewertungskriterien orientieren sich an der ISO-Norm 25010, welche Qualitäts-Kriterien für Software beschreibt [10] (Abbildung 1.1). Konkret werden die Software-Lösungen hinsichtlich folgender Bewertungskriterien untersucht:

1.4.1 Functional Suitability

- Erfüllt die Software die gestellten Anforderungen?

1.4.2 Compatibility

- Kann die Software mit vorhandenen Systemen interagieren?
- Entstehen Probleme durch den Einsatz der Software?
- Kann die Software mit ähnlichen Systemen koexistieren?

1.4.3 Usability

- Wie leicht ist der Umgang mit der Software zu erlernen?
- Wie leicht ist die Software bedienbar?
- Wie gut ist die Software gegen Nutzer-Fehler geschützt?
- Wie ansprechend ist die Benutzer-Oberfläche gestaltet?
- Wie barrierefrei ist die Software?

1.4.4 Security

- Wie vertraulich agiert die Software?
- Sind Verantwortlichkeiten nachvollziehbar?
- Wie geschieht die Authentifizierung?

1.4.5 Maintainability

- Lassen sich Komponenten der Software wiederverwenden?
- Wie leicht lässt sich die Software verändern?
- Lässt sich die Software testen?

1.4.6 Portability

- In welchen System-Umgebungen kann die Software eingesetzt werden?

2 Betrachtete Software-Lösungen

2.1 Ansible

2.1.1 Allgemein

Ansible ist eine Open Source Automatisierungs-Plattform von *Red Hat, Inc.* zum automatisierten Ausrollen von Konfigurationen (im Folgenden: *Playbook*s) über das Netzwerk.

2.1.2 Begriffsbildung

Ansible bezeichnet zunächst lediglich den Kern der Automatisierungs-Plattform, welcher traditionell über die Kommandozeile genutzt wird. Aus Gründen der Bedienbarkeit wird im Folgenden jedoch *AWX*, ein Web-Interface für *Ansible*, betrachtet und synonym zu *Ansible* verwendet. *AWX* stellt dabei das Open Source Upstream Projekt für das kommerzielle Produkt *Ansible Automation Controller* dar, welches zuvor unter dem Namen *Ansible Tower* bekannt war.

2.1.3 Verteilung

Abbildung 2.1 zeigt, dass *AWX* mit Hilfe eines Operators in einem Kubernetes-Cluster installiert wird. Die daraus entstehenden Kubernetes-Pods nehmen die Rolle des steuernden Servers (im Folgenden: *Control Node*) ein. Per *Secure Shell (SSH)* findet die Kommunikation zwischen der *Control Node* und dem zu steuernden System (im Folgenden: *Managed Node*) statt. Zusätzlich muss auf der *Managed Node Python* installiert sein. Ein User, wie beispielsweise ein System-Administrator, kann sich per Browser mit dem Web-Interface verbinden, um die *Control Node* zu bedienen.

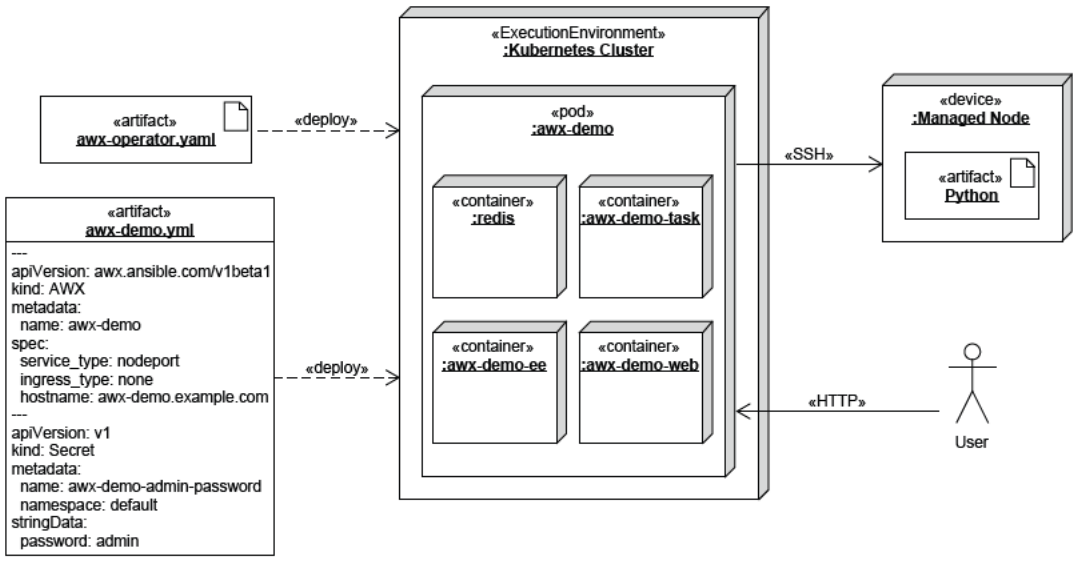

Abbildung 2.1: Ansible - AWX Verteilungs-Diagramm

In der Vergangenheit bestand auch die Möglichkeit, *AWX* in einem Docker-Container zu installieren. Dies wird inzwischen allerdings nicht mehr offiziell unterstützt.

2.1.4 Installation

```
# install Docker
sudo zypper --non-interactive install docker
sudo systemctl enable docker
sudo systemctl start docker

# add current user to docker group
sudo usermod -aG docker $USER

# install Minikube
curl -LO https://storage.googleapis.com/\
minikube/releases/v1.22.0/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube

# start Minikube cluster with privileges of newly joined group
/usr/bin/newgrp docker <<EONG
    minikube start --addons=ingress --cpus=4 --cni=flannel \
        --install-addons=true --kubernetes-version=stable --memory=6g
EONG

# create kubectl alias
alias kubectl="minikube kubectl --"
```

Quellcode 1: Ansible - AWX - Kubernetes-Cluster auf lokaler Maschine bereitstellen

Quellcode 1 zeigt, wie zu Testzwecken auf einem einzelnen System ein Kubernetes-Cluster eingerichtet werden kann.

```
# deploy awx operator into minikube cluster
kubectl apply -f https://raw.githubusercontent.com/\
ansible/awx-operator/0.13.0/deploy/awx-operator.yaml

# wait for deployment of operator to finish
kubectl rollout status deployment awx-operator

# create config file
cat > ~/awx-demo.yml <<EOF
---
apiVersion: awx.ansible.com/v1beta1
kind: AWX
metadata:
  name: awx-demo
spec:
  service_type: nodeport
  ingress_type: none
  hostname: awx-demo.example.com
---
apiVersion: v1
kind: Secret
metadata:
  name: awx-demo-admin-password
  namespace: default
stringData:
  password: admin
EOF

# apply config
kubectl apply -f awx-demo.yml

# wait a moment for the operator to start working
sleep 30

# wait for deployment of actual awx pod
kubectl rollout status deployment awx-demo

# forward port 8080 of current host to port 80 of awx pod
kubectl port-forward --address 0.0.0.0 service/awx-demo-service 8080:80
```

Quellcode 2: Ansible - Ausrollen des AWX Operators

Anschließend kann Quellcode 2 gefolgt werden, um den AWX-Operator auf dem vorhandenen Kubernetes-Cluster auszurollen und mit Hilfe der dort erstellten *awx-demo.yml* die eigentliche Installation von *AWX* anzustoßen. Sind alle Schritte beendet, ist das Web-

Interface über Port 8080 zu erreichen und die Anmeldung per Benutzernamen *admin* mit dem Passwort *admin* möglich.

2.1.5 Sprache

*Playbook*s werden in *YAML*, einer menschenlesbaren Auszeichnungssprache, verfasst, indem vordefinierte Funktions-Einheiten (im Folgenden: *Module*s) angegeben und eventuell notwendige Parameter (im Folgenden: *Module Argument*) spezifiziert werden. Somit ergibt sich eine Abstraktions-Ebene, welche keine Programmier-Kenntnisse voraussetzt [3]. Für komplexere Aufgaben lassen sich auch eigene *Module*s entwickeln, die in einer beliebigen Sprache formuliert sein können. Es wird jedoch empfohlen, *Python* zu nutzen, da hier bereits nützliche Libraries zur Verfügung stehen [4].

2.1.6 Grundlegende Funktionsweise

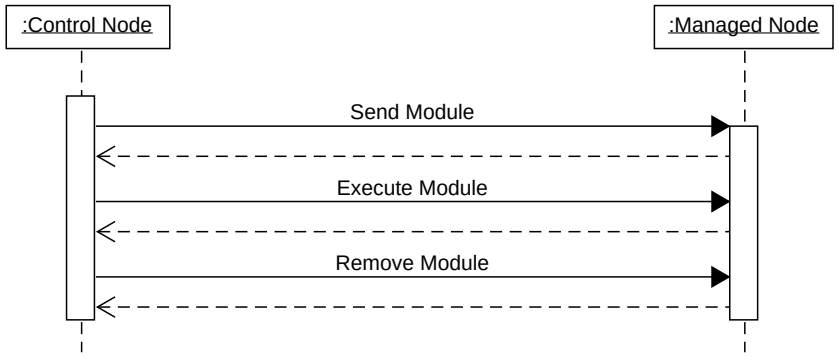

Abbildung 2.2: Ansible - Ausführung eines Modules

Wie in Abbildung 2.2 dargestellt, nutzt *Ansible* den sogenannten *Push Mode* [5]. Dies bedeutet, dass der steuernde Server (im Folgenden: *Control Node*) das auszuführende *Module* zunächst auf die *Managed Node* kopiert, dieses ausführt und im Anschluss wieder entfernt. Somit kann ein *Playbook* quasi in Echtzeit angewendet werden und es verbleibt kein Code auf der *Managed Node* [1][5]. Alternativ bietet *Ansible* auch einen *Pull Mode* an, bei welchem ein Dienst auf der *Managed Node* installiert wird, welcher in regelmäßigen Abständen nach ausstehenden Aufgaben sucht [6]. Im Folgenden wird der *Pull Mode* im Zusammenhang mit *Ansible* jedoch nicht weiter betrachtet.

2.1.7 Konzeptioneller Aufbau

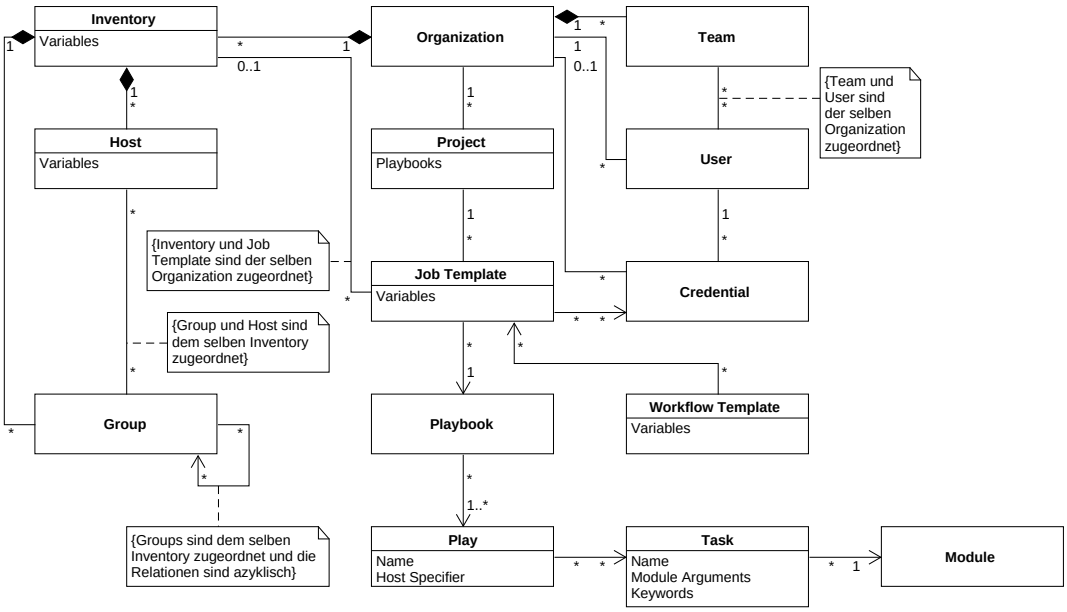

Abbildung 2.3: Ansible - AWX Klassen-Diagramm

Organization

Die *Organization* stellt in *AWX* die höchste Einheit in der Organisations-Hierarchie dar und dient als Container für *Users*, *Teams*, *Projects* und *Inventory*s.

User

User modelliert den tatsächlichen Benutzer, welcher mit der Web-Oberfläche von *AWX* interagiert. Nutzer können entweder per Oberfläche angelegt oder auch von anderen Systemen, besipielsweise per LDAP, importiert werden. Bei der Erstellung über das Web-Interface muss ein User exakt einer Organization zugeordnet werden.

Team

Ein Team stellt eine Sammlung von Usern dar. Ein User kann beliebig vielen Teams zugeordnet werden, wobei das Team und der User zur selben Organization gehören müssen.

Credential

Credentials bieten die Möglichkeit, Authentifizierungs-Informationen zentral auf dem *Ansible*-Server abzulegen. Beispielsweise können dies Anmelde-Informationen für die *SSH*-Verbindung mit dem Zielsystem oder auch Tokens für die Authentifizierung bei Versionierungs-Systemen wie *Git* sein. Credentials sind zunächst nur für den erstellenden Benutzer nutzbar, können aber auch mit der Organization geteilt werden. Somit kann anderen die Authentifizierung ermöglicht werden, ohne die eigentlichen Informationen preisgeben zu müssen.

Host

Ein *Host*, oder auch *Managed Node*, repräsentiert ein zu steuerndes System. *Host*s sind Teil von genau einem *Inventory*. Zusätzlich können *Host*-spezifische Variablen zur Nutzung in *Playbook*s definiert werden.

Inventory

Sammlung von *Host*s und genau einer *Organization* zugeordnet.

Group

Erlaubt das Bilden von Teilmengen von *Host*s innerhalb eines *Inventory*s. Eine *Group* kann andere *Groups* enthalten, jedoch muss die Relation azyklisch bleiben.

Project

Genau einer *Organization* zugeordnet. Zeigt auf ein Verzeichnis mit *Playbook*s und stellt diese für *Job Templates* zur Verfügung.

Job Template

Verweist auf genau ein *Playbook* des zugeordneten *Projects* und optional auf ein *Inventory*.

Workflow Template

Kann mehrere *Job Templates* des selben *Projects* verwenden, um einen Workflow darzustellen. *Job Templates* können mit Hilfe von drei Bedingungen - *On Success*, *On Fail* und *Always* - miteinander verknüpft werden.

Playbook

Datei für die Verhaltens-Modellierung. Verfasst in *YAML*. Kann ein oder mehrere *Play*s enthalten.
Quellcode 3 zeigt ein beispielhaftes *Playbook*, welches das in Abbildung 2.4 dargestellte Verhalten aufweist.

Falls die Datei *abc.txt* existiert, wird diese gelöscht, falls sie nicht existiert, wird sie erstellt. Anschließend wird *"Hello World"* ausgegeben, falls der ausführende *Host* Mitglied der *Group webservers* ist.
Effektiv wechselt also mit jeder Ausführung des *Playbook*s der Zustand der Datei *abc.txt* zwischen Existenz und Nicht-Existenz.

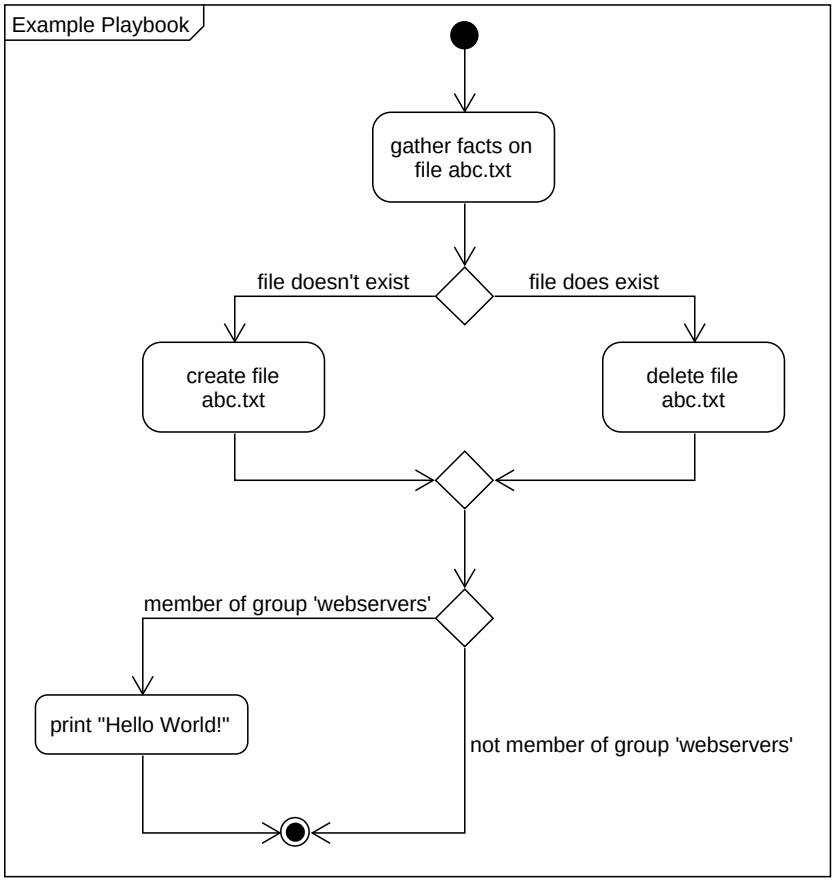

Abbildung 2.4: Ansible - Example Playbook Aktivitäts-Diagramm

```
1  - name: First Play with multiple Tasks
2    hosts: all
3    tasks:
4      - name: Task 1 of first Play
5        stat:
6          path: abc.txt
7        register: st
8
9      - name: Task 2 of first Play
10       shell: touch abc.txt
11       when: st.stat.isreg is not defined or st.stat.isreg == False
12
13     - name: Task 3 of first Play
14       shell: rm frettchen.txt
15       when: st.stat.isreg is defined and st.stat.isreg == True
16
17 - name: Will only be executed on hosts within Group 'webservers'
18   hosts: webservers
19   tasks:
20     - name: Task 1 of second Play
21       debug:
22         msg: "Hello World!"
```

Quellcode 3: Ansible - examplePlaybook.yml

Play

Ein *Play* ist Teil eines *Playbook*s, kann einen Namen besitzen, spezifiziert mit Hilfe des *hosts*-Keywords die Teil-Menge der *Host*s, auf welchen es ausgeführt werden soll und liefert eine Menge von *Task*s. Teil eines *Playbook*s.

Task

Verweist auf ein *Module* und liefert Parameter für dieses. Kann außerdem Keywords wie *when* enthalten, um die Ausführung beispielsweise mit einer Kondition zu verknüpfen. Das Keyword *register* erlaubt es, Rückgabewerte von *Module*s zu speichern und in nachfolgenden *Task*s zu referenzieren. Teil eines *Play*s.

Module

Kleinste funktionale Einheit innerhalb von *Ansible*. Stellt tatsächliches Verhalten dar. Vorrangig verfasst in *Python*. Parameter werden durch den aufrufenden *Task* geliefert.

2.2 Chef

2.2.1 Allgemein

Chef ist eine Open Source Automatisierungs-Plattform von *Progress Software Corporation* zum automatisierten Ausrollen von Konfigurationen (im Folgenden: *Run-List*s) über das Netzwerk, welche als *Infrastructure as Code* Software beschrieben wird.

2.2.2 Begriffsbildung

Der Begriff *Chef* betitelt zunächst das gesamte *Chef*-Ökosystem. Im Rahmen dieser Arbeit werden jedoch hauptsächlich *Chef Infra*, *Chef Workstation* und *Chef Infra Client* betrachtet. Als Web-Interface findet außerdem das kostenpflichtige *Chef Automate* Anwendung.

2.2.3 Verteilung

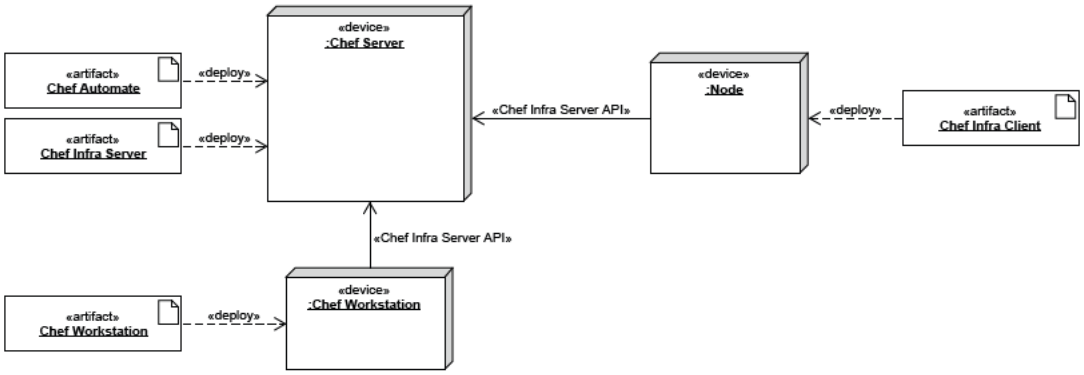

Abbildung 2.5: Chef - Verteilungs-Diagramm

Wie Abbildung 2.5 zu entnehmen ist, werden die Server-Dienste *Chef Infra Server* und *Chef Automate* direkt auf dem System installiert, welches die Rolle des steuernden Servers (im Folgenden: *Chef Server*) übernehmen soll. Das Erstellen der Konfigurationen (im Folgenden: *Cookbooks*) geschieht über ein separates System, die *Chef Workstation*, auf welchem der gleichnamige Dienst installiert ist. Auf den zu konfigurierenden Systemen (im Folgenden: *Nodes*) muss der *Chef Infra Client* installiert sein, sodass diese regelmäßig über die *Chef Infra Server API* bei dem *Chef Server* nach ausstehenden Konfigurationen anfragen.

2.2.4 Installation

Chef bietet ein Skript an, welches die Installation, wie in Quellcode 4 dargestellt, sehr einfach hält.

```
1 curl https://packages.chef.io/files/current/latest/\
2 chef-automate-cli/chef-automate_linux_amd64.zip |
3 gunzip - > chef-automate && chmod +x chef-automate
4
5 sudo sysctl -w vm.max_map_count=262144
6 sudo sysctl -w vm.dirty_expire_centisecs=20000
7
8 sudo ./chef-automate deploy --product automate --product infra-server
```

Quellcode 4: Chef - Installation von *Chef Automate* und *Chef Infra*

2.2.5 Sprache

Um die gewünschte Konfiguration einer *Node* zu beschreiben, wird dieser eine *Run-List* zugeordnet. Eine *Run-List* ist eine Liste von *Ruby*-Dateien, welche den gewünschten Konfigurations-Stand beschreiben (im Folgenden: *Recipe*s). In den *Recipe*s besteht die Möglichkeit, neben reinem *Ruby* auch sogenannte *Resource*s zu verwenden. Dies sind vorgefertigte Funktions-Einheiten, welche die Durchführung von häufig ausgeführten Aufgaben vereinfachen sollen. Dennoch ist eine gewisse Vertrautheit mit *Ruby* empfehlenswert. Alternativ können *Recipe*s allerdings auch in *YAML* formuliert werden.

2.2.6 Grundlegende Funktionsweise

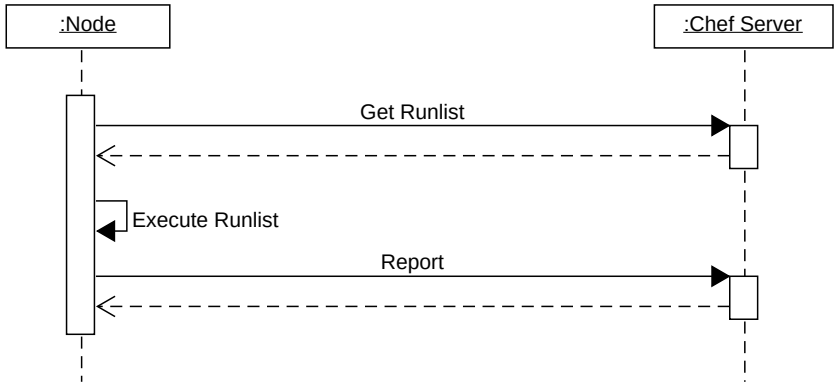

Abbildung 2.6: Chef - Ausführen einer Run-List

Abbildung 2.6 zeigt stark vereinfacht, dass die *Node* ihre *Run-List* von dem *Chef Server* bezieht, diese abarbeitet und anschließend den Status an den *Chef Server* zurückmeldet. Für eine detailliertere Darstellung siehe Literaturverweis [18].

2.2.7 Konzeptioneller Aufbau

Organization

Die höchste Organisations-Einheit von *Chef*. Container für *Groups*, *User*s, *Node*s und *Cookbook*s.

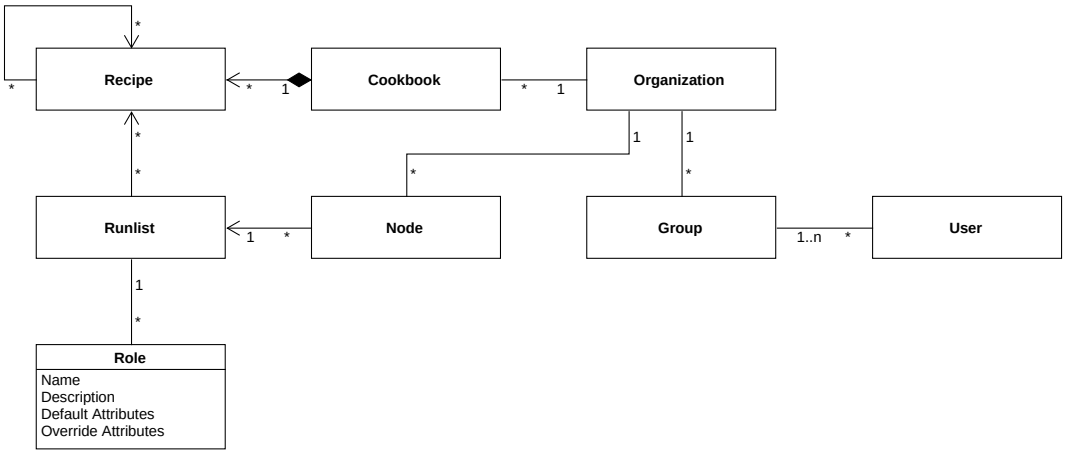

Abbildung 2.7: Chef - Klassen-Diagramm

Group

Gehört zu exakt einer *Organization* und stellt einen Container für *User*s dar.

User

Modelliert einen Nutzer, welcher mit *Chef Infra* interagiert. Ein *User* kann Mitglied mehrerer *Groups* sein.

Cookbook

Ordner-Struktur, welche unter anderem *Recipe*s enthält. Ein *Cookbook* gehört immer zu genau einer *Organization*.

Recipe

Konfigurations-Datei in *Ruby* oder *YAML* verfasst. Beschreibt den angestrebten Konfigurations-Stand mit Hilfe von *Resource*s. Ein *Recipe* kann andere *Recipe*s enthalten und von *Run-List*s referenziert werden.
Quellcode 5 zeigt ein beispielhaftes *Recipe*, welches das in Abbildung 2.8 dargestellte Verhalten aufweist.

Falls die Datei *abc.txt* nicht existiert, wird diese erstellt. Daraufhin wird geprüft, ob der Inhalt der Datei dem String *Hallo* entspricht. Ist dies nicht der Fall, wird der Inhalt der Datei angepasst. Anschließend wird mit dem Log-Level Info der String *Hello World* geloggt.
Nach der Ausführung des *Recipe*s ist also sichergestellt, dass die Datei *abc.txt* mit dem Inhalt *Hallo* existiert.

```
1 file "#{ENV['HOME']}/abc.txt" do
2   content 'Hallo'
3 end
4
5 log "Hello World!" do
6   level :info
7 end
```

Quellcode 5: Chef - exampleRecipe.rb

Run-List

Liste von anzuwendenden *Recipe*s und *Role*s, welche einer *Node* zugeordnet werden kann.

Node

Ein durch den *Chef Server* zu konfigurierendes System. Gehört zu genau einer *Organization* und hat exakt eine *Run-List*.

Role

Run-List mit Metadaten, welche wiederum Teil einer *Run-List* sein kann. Erlaubt es somit, dass eine *Run-List* eine andere referenziert.

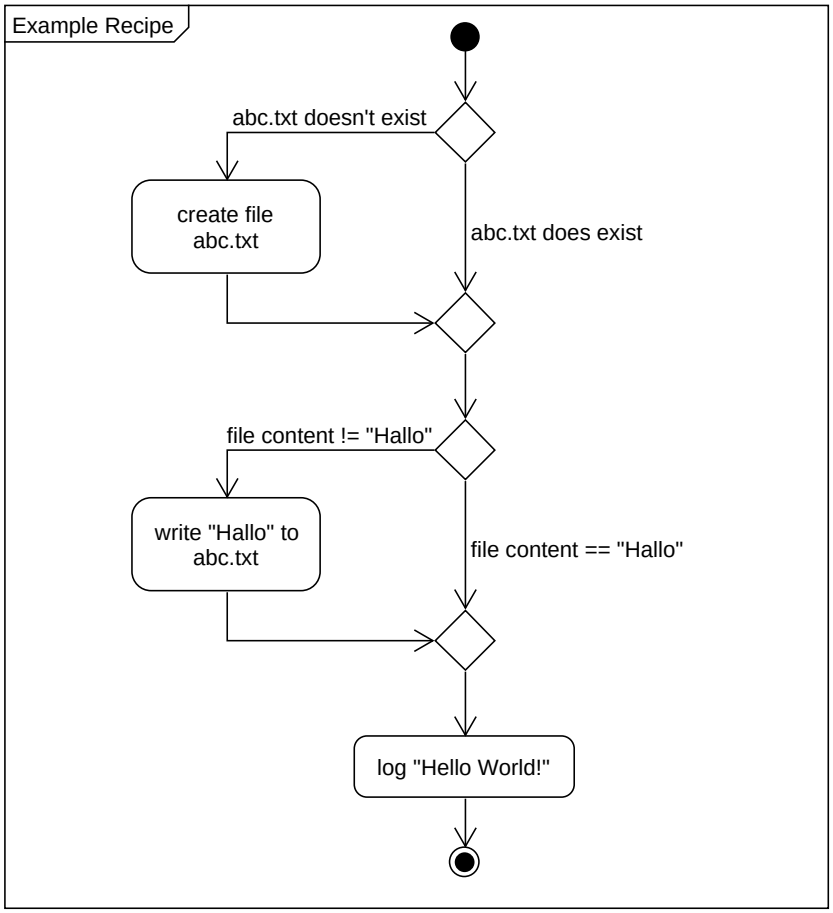

Abbildung 2.8: Chef - Example Recipe Aktivitäts-Diagramm

3 Fallstudien

Im Folgenden werden Anwendungsfälle untersucht und mögliche Umsetzungen beider Software-Lösungen gegenübergestellt.

3.1 Einrichten eines Web-Servers

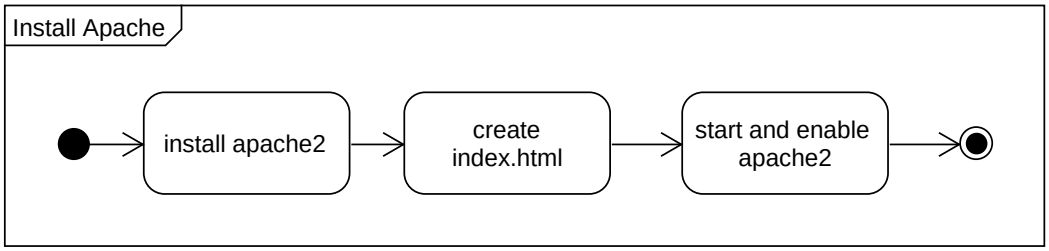

Abbildung 3.1: Install Apache Aktivitäts-Diagramm

Aufgabe innerhalb dieser Fallstudie ist es, einen Apache-Webserver auf einem Debian-basierten System (Package-Manager apt) einzurichten. Dafür müssen die in Abbildung 3.1 dargestellten Schritte durchlaufen werden.

3.1.1 Ansible

```yaml
- name: set up simple webserver
  hosts: all
  tasks:
    - name: make sure apache is installed
      apt:
        name: apache2
        state: present
    - name: create static test page
      copy:
        content: "Dies ist eine statische Test-Seite"
        dest: /var/www/html/index.html
        owner: 'www-data'
        group: 'www-data'
    - name: make sure apache is running and will start on reboot
      service:
        name: apache2
        enabled: yes
        state: started
```

Quellcode 6: Ansible - installApache.yml

3.1.2 Chef

```
apt_package 'apache2' do
  action :install
end

file '/var/www/html/index.html' do
  content "Dies ist eine statische Test-Seite"
  owner 'www-data'
  group 'www-data'
  action :create
end

service 'apache2' do
  action [ :enable, :start ]
end
```

Quellcode 7: Chef - installApache.rb

3.1.3 Auswertung

Sowohl *Chef*, als auch *Ansible* bieten Mechanismen an, um Abbildung 3.1 zu folgen. Auch wird die Ähnlichkeit zwischen *Module*s und *Resource*s dadurch verdeutlicht, dass die genutzten Parameter nahezu identisch sind.
Da es sich bei dieser Fallstudie um eine sehr grundlegende Aufgabe handelt, ist das Ergebnis nicht verwunderlich.

3.2 Automatisches Erneuern von SSL-Zertifikaten

Im Folgenden besteht die Aufgabe, automatisiert *Secure Sockets Layer (SSL)* Zertifikate zu generieren bzw. zu erneuern, falls diese abgelaufen sind. Zur Veranschaulichung wird eine stark abstrahierte Version des *Automatic Certificate Management Environment (ACME)* Protokolls [23] verwendet, deren Funktionsweise durch Abbildung 3.2 beschrieben wird.

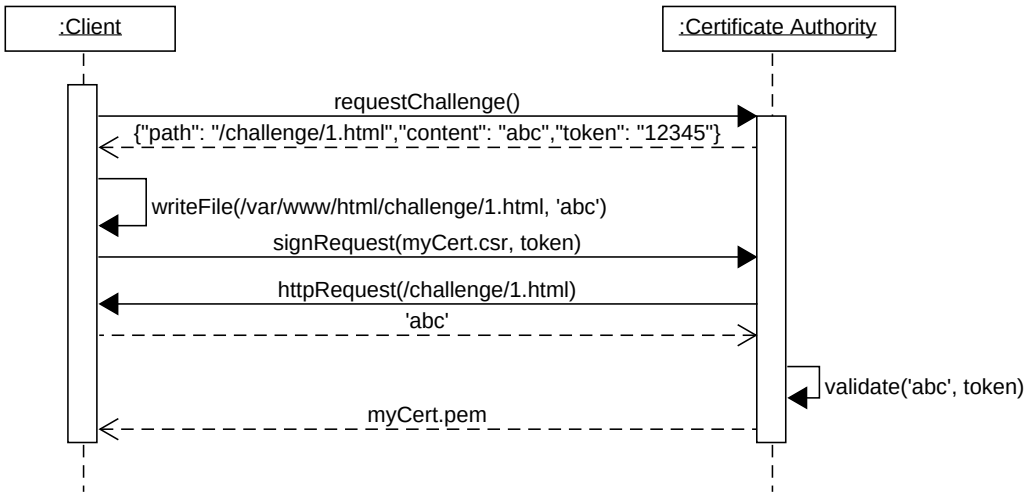

Abbildung 3.2: ACME-Abstraktion Aktivitäts-Diagramm

So fragt eine Partei, welche ein unterschriebenes Zertifikat anfordern möchte (im Folgenden: *Client*), bei der *Certificate Authority* eine Aufgabe an, um ihre Identität zu bestätigen (im Folgenden: *Challenge*). In Abbildung 3.2 ist die Challenge beispielsweise so formuliert, dass in dem Pfad */challenge/* eine Datei *1.html* mit dem Inhalt *abc* erstellt

werden soll. Nachdem der *Client* die entsprechende Datei angelegt hat, sendet dieser ein *Certificate Signing Request*, zusammen mit dem zuvor übergebenen Token. Anschließend validiert die *Certificate Authority* die korrekte Umsetzung der *Challenge* und sendet gegebenenfalls das unterschriebene Zertifikat zurück. Nachfolgend wird der Prozess in der Rolle des *Clients* im Austausch mit der fiktiven *Certificate Authority* mit dem Namen *sign-my-cert.com* modelliert.

3.2.1 Ansible

```yaml
- name: generate and sign certificate
  hosts: all
  tasks:
    - name: get certificate information
      community.crypto.x509_certificate_info:
        path: myCert.pem
        valid_at:
          one_day: "+1d"
      register: certinfo
      ignore_errors: yes

    - name: delete private key if certificate expires in less than 24 hours
      file:
        path: "{{ item }}"
        state: absent
      with_items:
        - myCert.key
      when: certinfo.valid_at is defined and not certinfo.valid_at.one_day

    - name: generate private key
      community.crypto.openssl_privatekey:
        path: myCert.key

    - name: generate signing request
      community.crypto.openssl_csr:
        path: myCert.csr
        common_name: abc.de
        privatekey_path: myCert.key
        organization_name: ABC Corporation
        organizational_unit_name: A
        country_name: DE
      notify: "send signing request"
```

Quellcode 8: Ansible - renewCertificate.yml

```
33  handlers:
34    - name: request challenge
35      uri:
36        url: https://sign-my-cert.com/getChallenge.php
37        method: GET
38        return_content: yes
39        status_code: 200
40        body_format: json
41      register: challenge
42      listen: "send signing request"
43
44    - name: write content of challenge to the desired path
45      copy:
46        content: "{{ challenge.json.content }}"
47        dest: "/var/www/html/{{ challenge.json.path }}"
48      listen: "send signing request"
49
50    - name: send token and signing request to receive certificate
51      uri:
52        url: https://sign-my-cert.com/sign.php
53        method: POST
54        return_content: yes
55        status_code: 200
56        body_format: form-urlencoded
57        body:
58        - [ token, "{{ challenge.json.token }}" ]
59        - [ csr, "{{ lookup('file','myCert.csr') }}" ]
60      register: signresponse
61      listen: "send signing request"
62
63    - name: write signed certificate to myCert.pem
64      copy:
65        content: "{{ signresponse.content }}"
66        dest: myCert.pem
67      listen: "send signing request"
```

Quellcode 9: Ansible - installApache.yml Fortsetzung

Der Lösungs-Ansatz aus Quellcode 8 und Quellcode 9 führt zu dem in Abbildung 3.3 dargestellten Aktivitäts-Diagramm.

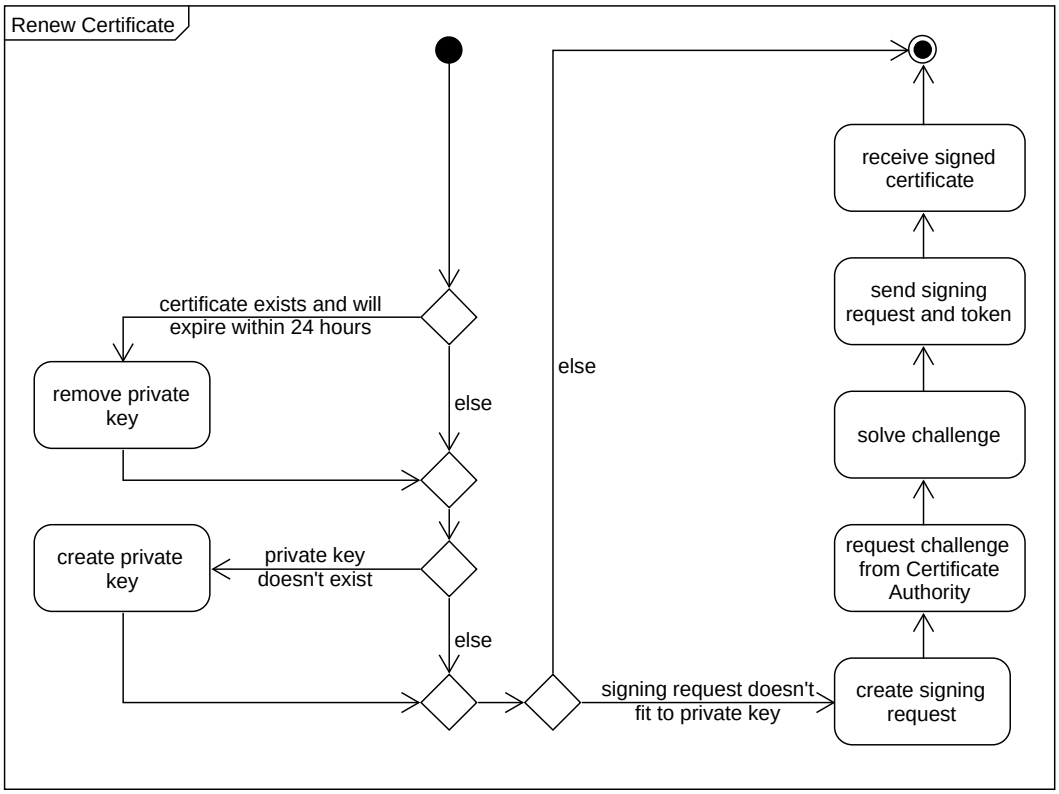

Abbildung 3.3: Ansible - Renew Certificate Aktivitäts-Diagramm

3.2.2 Chef

```
path = "#{ENV['HOME']}"
wwwroot = "/var/www/html"

execute "remove expired certificates" do
  command "rm #{path}/myCert.*"
  not_if "test -f #{path}/myCert.pem && openssl x509 -checkend 86400 \
  -noout -in #{path}/myCert.pem"
end

openssl_x509_request "#{path}/myCert.csr" do
  common_name 'abc.de'
  org 'ABC Corporation'
  org_unit 'A'
  country 'DE'
  key_type 'rsa'
end

ruby 'sign newly created certificate' do
  code <<-EOH
    require 'net/http'
    require 'json'

    uri = URI('https://sign-my-cert.com/getChallenge.php')
    response = Net::HTTP.get(uri)
    challenge = JSON.parse(response)

    signingRequest = File.read("myCert.csr")
    File.open("#{wwwroot}/challenge['path']" , "w") \
    { |f| f.write challenge['content'] }

    uri = URI('https://sign-my-cert.com/sign.php')
    res = Net::HTTP.post_form(uri, 'csr' => 'signingRequest', \
    'token' => challenge['token'])

    File.open("myCert.pem" , "w") { |f| f.write res.body }
  EOH
  cwd path
  action :nothing
  subscribes :run, "openssl_x509_request[#{path}/myCert.csr]", :immediately
end
```

Quellcode 10: Chef - renewCertificate.rb

Das Verhalten von Quellcode 8 ist in Abbildung 3.4 beschrieben.

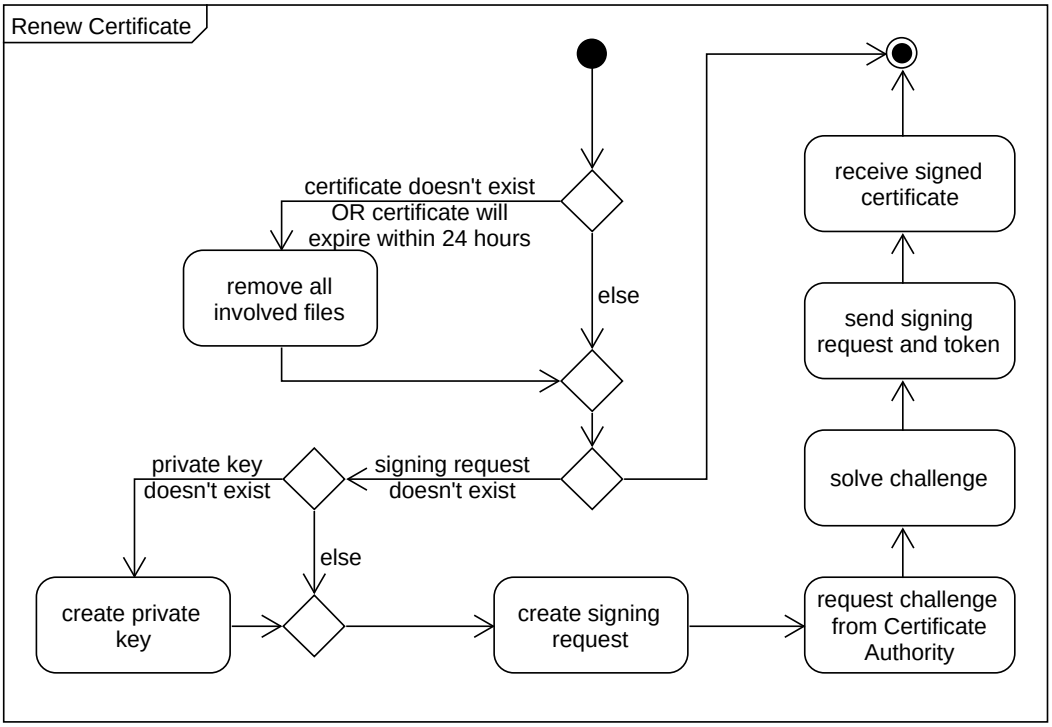

Abbildung 3.4: Chef - Renew Certificate Aktivitäts-Diagramm

3.2.3 Auswertung

Beide Software-Lösungen haben die notwendigen Mittel, um den beispielhaften Austausch umzusetzen.

Während der Lösungs-Ansatz bei *Ansible* deutlich länger ausfällt, ist man bei der Benutzung von *Chef* dazu gezwungen, die Abstraktions-Ebene der *Resource*s zu verlassen und auf reinen *Ruby*-Code auszuweichen. Dies ist vor allem dadurch bedingt, dass *Resource*s im Gegensatz zu den *Module*s von *Ansible* keinen Rückgabewert besitzen und so nur ein sehr eingeschränkter Austausch von Informationen zwischen *Resource*s stattfinden kann.

Ähnlichkeiten zeigen sich vor allem in den Signalisierungs-Mechanismen. So ähnelt *Ansible*s Konzept von *notify* und *listen* sehr dem *subscribes*-Ansatz von *Chef*.

Warnend hervorzuheben ist, dass *Chef*s *openssl_x509_request-Resource* lediglich anhand der Anwesenheit der Ziel-Datei prüft, ob eine Aktion erforderlich ist. Anders als bei

Ansibles openssl_ csr-Module garantiert diese nicht, dass der *Certificate Signing Request* zu dem referenzierten *Private Key* passt. Ohne vorige Abfrage könnte es also passieren, dass ein *Certificate Signing Request* versendet wird, zu welchem der zugehörige *Private Key* nicht mehr existiert. Das vollständige *ACME*-Protokoll bietet zwar einen weiteren Verifizierungs-Schritt, um exakt dieser Problematik zu begegnen, jedoch sollte dieser Umstand an dieser Stelle Erwähnung finden.

Abseits der zuvor genannten Punkte und leichten Unterschiede der Aktivitäts-Diagramme sind beide beide Lösungs-Ansätze als gleichwertig zu betrachten.

3.3 Sequenzielles Herunterfahren von Diensten auf verschiedenen Systemen

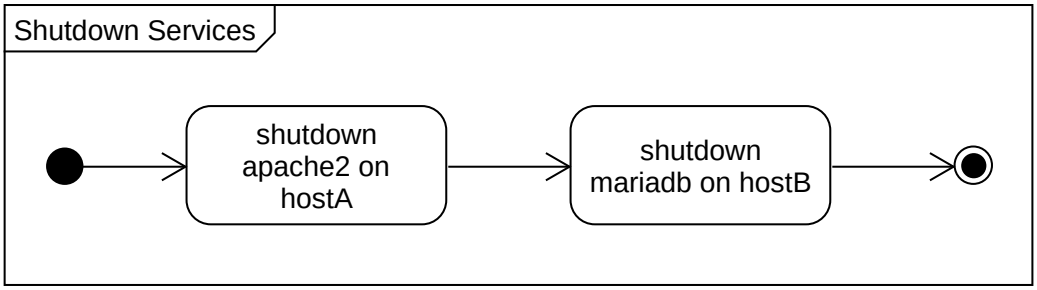

Abbildung 3.5: Shutdown Services Aktivitäts-Diagramm

Ziel ist das Herunterfahren von Diensten in einer festgelegten Reihenfolge. So soll in dem durch Abbildung 3.5 beschriebenen Beispiel erst der Web-Server auf *hostA* beendet werden, bevor die Datenbank auf *hostB* abgeschaltet wird.

3.3.1 Ansible

```
- name: shut down webserver on hostA
  hosts: hostA
  tasks:
    - name: make sure apache is stopped
      service:
        name: apache2
        state: stopped

- name: shut down database on hostB
  hosts: hostB
  tasks:
    - name: make sure mariadb is stopped
      service:
        name: mariadb
        state: stopped
```

Quellcode 11: Ansible - shutdownServices.yml

3.3.2 Auswertung

Bei der Benutzung von *Ansible* zeigt sich diese Aufgabe als trivial. Da *Task*s und *Play*s in der Standard-Konfiguration sequentiell ausgeführt werden, müssen die *Play*s, welche den jeweiligen Ziel-*Host* beschreiben, lediglich in der richtigen Reihenfolge positioniert werden.

Chef bietet, bedingt durch den verwendeten *Pull Mode*, hingegen nativ keine Möglichkeit, Aktionen auf verschiedenen *Node*s zueinander koordiniert auszuführen.

4 Diskussion

4.1 Auswertung

4.1.1 Functional Suitability

Erfüllt die Software die gestellten Anforderungen?

Ansible Erfüllt mehrere Anforderungs-Gebiete. Die Nutzung des *Push Mode*s erlaubt einerseits den Einsatz als reine Beschreibung von Infrastuktur, unterstützt andererseits aber auch komplexe Orchestrierung mehrerer *Host*s.

Chef Erfüllt lediglich das Anforderungs-Gebiet der Infrastruktur-Beschreibung. Komplexere Aufgaben sind schwierig bis gar nicht umsetzbar.

4.1.2 Compatibility

Kann die Software mit vorhandenen Systemen interagieren?

Ansible Komplexe Interaktion mit anderen Systemen sind besipeilsweise über *Hyper Text Transfer Protocol (HTTP)* Requests möglich.

Chef Eingeschränkte Interaktions-Möglichkeiten durch die Tatsache, dass *Resource*s keinen Rückgabewert liefern und somit ohne die Nutzung von reinen *Ruby*-Scripts nur Interaktionen erlauben, die uni-direktional sind.

Entstehen Probleme durch den Einsatz der Software?

Ansible Im Rahmen der Untersuchung waren keine Probleme ersichtlich.

Chef Im Rahmen der Untersuchung waren keine Probleme ersichtlich.

Kann die Software mit ähnlichen Systemen koexistieren?

Ansible Im Rahmen der Untersuchung wurden keine Unverträglichkeiten aufgedeckt. *Progress Software Corporation* wirbt damit, dass *Chef InSpec*, ein anderes Produkt aus dem *Chef*-Ökosystem, eine gute Ergänzung zum Konfigurations-Management mit *Ansible* darstellt [21].

Chef Im Rahmen der Untersuchung wurden keine Unverträglichkeiten aufgedeckt. *Progress Software Corporation* wirbt damit, dass *Chef InSpec*, ein anderes Produkt aus dem *Chef*-Ökosystem, eine gute Ergänzung zum Konfigurations-Management mit *Ansible* darstellt [21].

4.1.3 Usability

Wie leicht ist der Umgang mit der Software zu erlernen?

Ansible Die Abstraktions-Ebene, welche durch *Playbook*s gegeben ist, zusammen mit einer Vielzahl an bereits vorhandenen, sehr spezifischen *Module*s, bietet einen einfachen Einstieg und ermöglicht das Verfassen von komplexen Anweisungen - auch mit wenig bis gar keiner Programmier-Erfahrung. Die eigentliche Ausführung von bereits verfassten *Playbook*s kann durch einen einzelnen Maus-Klick angestoßen werden und stellt somit keinerlei Lern-Hürde dar.

Chef Verlangt schnell zumindest rudimentäre Erfahrung in *Ruby*. Komplexere Aufgaben sind nur mit guten Programmier-Kenntnissen zu bewältigen. Interaktion erfordert Kenntnisse im Umgang mit der Kommando-Zeile. Leider ist die offizielle Dokumentation voll mit Syntax-Fehlern, was den Lernfluss unnötig erschwert.

Wie leicht ist die Software bedienbar?

Ansible Zusammen mit der Web-Oberfläche von *AWX* bietet *Ansible* eine äußerst intuitive und einfache Bedien-Oberfläche, sobald die grundlegenden Konzepte verinnerlicht wurden. Dass *AWX* von jedem System aus, welches einen Web-Browser besitzt, bedient werden kann, erleichtert die Bedienung noch weiter.

Chef Zwar bietet *Chef Automate* ebenfalls eine Web-Oberfläche an, jedoch agiert diese lediglich als Dashboard und zeigt für jede *Node* nur den Status des aktuellsten Durchlaufs an. Die Bedienung muss von einem System, auf welchem *Chef Workstation* installiert ist, über die Kommandozeile geschehen und ist somit fortgeschrittenen Nutzern vorbehalten.

Wie gut ist die Software gegen Nutzer-Fehler geschützt?

Ansible Innerhalb der Web-Oberfläche sind Bedienfehler durch die strikte Struktur praktisch nicht möglich. Sollte versucht werden, eine ungültige Aktion auszuführen, erscheint eine Fehler-Meldung. Die Ausführung eines *Playbook*s wird standardmäßig automatisch für den *Host* abgebrochen, für den ein unbehandelter Fehler in einem *Task* aufgetreten ist.

Chef Die Ausführung fehlerhafter *Recipe*s wird abgebrochen, sobald eine *Resource* einen Fehler aufdeckt.

Wie ansprechend ist die Benutzer-Oberfläche gestaltet?

Ansible Sehr ansprechende und übersichtliche Web-Oberfläche.

Chef Lediglich Interaktion mit der Kommando-Zeile.

Wie barrierefrei ist die Software?

Ansible Durch die Web-Oberfläche von *AWX* können handelsübliche Screen-Reader zur Unterstützung bei der Bedienung eingesetzt werden können.

Chef Keine besonderen Accessibility-Features, da die Interaktion über die Kommando-Zeile geschieht und Auto-Vervollständigung fehlt.

4.1.4 Security

Wie vertraulich agiert die Software?

Ansible Kommunikation zwischen *Control Node* und *Managed Node* geschieht per *SSH*. Durch *Credentials* können anderen Nutzern sensitive Daten zur Nutzung bereitgestellt werden, ohne diese preisgeben zu müssen.

Chef Nutzung von *SSH*-Keys wird zur Authentifizierung von einer *Chef Workstation* beim *Chef Infra Server* erzwungen. Kommunikation zwischen *Node*s und *Chef Infra Server* ist über ein Server-Zertifikat geschützt.

Sind Verantwortlichkeiten nachvollziehbar?

Ansible Ja. Jede Ausführung eines *Templates*, inklusive Verantwortlichem Nutzer und dem spezifischen Output dieses Laufs, werden erfasst und sind in der Web-Oberfläche abrufbar.

Chef Nein, es ist nicht nachvollziehbar, wer die *Run-List* einer *Node* modifiziert hat.

Wie geschieht die Authentifizierung?

Ansible Authentifizierung mit *AWX* geschieht per Nutzername und Passwort. Nutzer können aus Fremdquellen wie beispielsweise LDAP importiert werden. Authentifizierung zwischen *Control Node* und *Managed Node* kann unter anderem wahlweise per Nutzername und Passwort oder auch per *SSH*-Key geschehen.

Chef Nutzer-spezifische *SSH*-Keys. Authentifizierung zwischen *Node* und *Chef Infra Server* geschieht per Server-seitigem Zertifikat.

4.1.5 Maintainability

Lassen sich Komponenten der Software wiederverwenden?

Ansible Ja, *Module*s, *Playbook*s und auch einzelne *Play*s lassen sich wiederverwenden. *Templates* bieten die Möglichkeit, *Playbook*s bei Bedarf auf verschiedenen *Inventory*s oder auch mit verschiedenen *Credentials* auszuführen.

Chef Ja, *Resource*s und *Recipe*s lassen sich wiederverwenden.

Wie leicht lässt sich die Software verändern?

Ansible *Module*s lassen sich einfach durch Hinzufügen in die Ordner-Hierarchie verfügbar machen.

Chef *Resource*s lassen sich einfach durch Hinzufügen in die Ordner-Hierarchie verfügbar machen.

Lässt sich die Software testen?

Ansible Ja, *Playbook*s können mit einer *check*-Konfiguration gestartet werden, welche Änderungen nur simuliert und nicht tatsächlich auf der *Managed Node* ausführt. Alternativ können natürlich auch *Inventory*s aus Test-Systemen zusammengestellt werden, um *Playbook*s in einer unkritischen Umgebung testen zu können.

Chef Ja, es gibt die Möglichkeit, automatisiert lokal auf der *Chef Workstation* eine virtuelle Test-Umgebung aufbauen zu lassen, um *Recipe*s zu testen.

4.1.6 Portability

In welchen System-Umgebungen kann die Software eingesetzt werden?

Ansible *AWX* braucht lediglich ein Kubernetes-Cluster um lauffähig zu sein. *Managed Node*s können auf Linux, Mac oder Windows laufen.

Chef *Chef Infra Server* benötigt Linux. *Node*s und *Chef Workstation*s können auf Linux, Mac oder Windows laufen.

4.2 Probleme

Nachfolgend einige Punkte, welche bei der Evaluation aufgefallen sind, aber nicht direkt Aufschluss über das zugrundeliegende System liefern.

4.2.1 Ansible

AWX nutzt nach der Installation kein SSL

Nach der Installation von *AWX* ist die Web-Oberfläche zunächst nur per *HTTP* zu erreichen. Eine Umstellung auf *SSL* ist nicht trivial und erfordert Kenntnisse im Umgang mit *Kubernetes*.
Als Folge davon ist die unveränderte Standard-Installation ein enormes Sicherheits-Risiko, da nicht nur die Anmelde-Daten des Nutzers, sondern vor allem auch die *Credentials* beim Eingeben auf der Web-Oberfläche völlig unverschlüsselt über das Netzwerk übertragen werden.

Nutzung von erhöhten Privilegien

Wird *Ansible* dazu angewiesen, ein *Playbook* als *sudo* auszuführen, kann es, bei falscher Konfiguration des Nutzers auf der *Managed Node*, dazu kommen, dass die Ausführung niemals endet. *Ansible* kann nämlich nicht erkennen, wenn das Betriebssystem nach einem Passwort für die Nutzung von *sudo* fragt.
Hier hilft nur die Anpassung der *visudo*-Datei, um dem betreffenden Nutzer zu erlauben, *sudo* ohne Passwort-Bestätigung auszuführen.

4.2.2 Chef

Rechtschreibfehler

Während der gesamten Dauer der Evaluation sind vermehrt Probleme aufgetreten, die sich auf Rechtschreibfehler innerhalb der *Chef*-Dokumentation zurückführen ließen.
Dies war schon beim *Quickstart-Guide* ein enormes Problem, da sich die textliche Repräsentation des Links für den Download des Installations-Pakets, von dem tatsächlich gefolgten Link unterschieden hat. Wurde der Link angeklickt, konnte das Paket heruntergeladen werden. Wurde der Link zur Nutzung auf der Kommandozeile kopiert, zeigte dieser auf eine Web-Ressource, die nicht existiert.
Auch Beispiele zur Nutzung von *Resource*s waren teilweise schlicht nicht funktionsfähig, da sie, beispielsweise durch ein verrutschtes Hoch-Komma, einen Syntax-Fehler enthielten.

Anmelde-Pflicht

Nach Durchführung des *Quickstart-Guides* ist zwar ein funktionsfähiges System installiert, jedoch fehlt die weitere Führung, um grundlegende Konzepte des Systems nachvollziehen zu können. Diese wird nur durch einen Online-Lehrgang geliefert, welcher einer Anmeldung bedarf und beinahe tägliche Werbe-Emails mit sich bringt.

5 Fazit

5.1 Bewertung

5.1.1 Ansible

Ansible, speziell zusammen mit *AWX*, ist ein sehr ausgereiftes, intuitives Produkt, welches leicht zu erlernen und zu bedienen ist. Es steht *Chef Infra* in nichts nach, da es statische Konfigurations-Szenarien vergleichbar souverän meistert. Zusätzlich erlaubt der *Push Mode* Steuerungs-Szenarien, welche mit *Chef* nur schwer oder sogar unmöglich umzusetzen wären. Dieser Ansatz ist außerdem sehr non-invasiv, da auf einer *Managed Node* keine Dienste installiert werden müssen und nach der Ausführung eines *Playbook*s kein Code auf dieser verbleibt. Features wie der Credential-Storage oder das tiefgehende Rechte-System machen es einfach und sicher, Administrations-Aufgaben mit anderen zu teilen. Durch die simple Web-Oberfläche lässt sich *Ansible* von praktisch überall verwenden und Administration bedenkenlos auch in die Hände von fachfremden Personen legen.

5.1.2 Chef

Chef Infra ist eine sehr fokussierte Software, welche auf exakt eine Aufgabe setzt: Beschreibung von Infrastruktur in Code. Dies zeigt sich zum einen durch das fehlende interaktive Web-Interface, zum anderen durch die Notwendigkeit der Interaktion durch *Chef Workstation*. Dass es keine Möglichkeit für eine *Resource* gibt, mit Werten aus anderen *Resource*s zu arbeiten, verhindert, dass innerhalb dieser Abstraktions-Ebene komplexere Automatisierung-Aufgaben umgesetzt werden können. Durch den genutzten *Pull Mode* entfällt auch die Möglichkeit der Steuerung im Gesamt-Kontext der Server-Flotte. Zusätzlich bedingt dieser die Installation von *Chef Infra Client*, damit die *Node* gesteuert

werden kann. *Chef* ist deutlich auf erfahrene System-Administratoren ausgerichtet und ist deswegen nicht für Anfänger oder fachfremde Personen zu empfehlen.

5.2 Empfehlung

Als Ergebnis der Untersuchung ist ganz klar *Ansible* zu empfehlen. Hauptsächlich durch die Nutzung von *Push Mode* kann es sämtliches Verhalten von *Chef* abbilden und noch mehr. Besonders für Anfänger ist es, wegen der einfachen Bedienung, zu bevorzugen.

5.3 Grenzen der Automatisierung

Ungeachtet der präferierten Automatisierungs-Plattform sollte immer kritisch untersucht werden, ob es Sinn macht, einen vorliegenden Arbeitsablauf zu automatisieren. Im Falle eines System, welches keine Programmier-Schnittstelle sondern nur eine Web-Oberfläche anbietet, könnte zum Beispiel die Automatisierung unter Zuhilfenahme von *Selenium* erwogen werden. Jedoch bleibt zu beachten, dass eine grafische Web-Oberfläche in der Regel nicht dazu ausgelegt ist, maschinell bedient zu werden und die Automatisierung somit Risiken der Fehlbedienung birgt, die im schlimmsten Fall nicht nachvollziehbar sind.
Ein besserer Ansatz wäre hier beispielsweise die Nutzung von *Ansible*s *Workflow Templates*, welche es erlauben, zwischen der Ausführung von zwei *Playbook*s auf Nutzer-Interaktion zu warten. So könnte der Nutzer zum Beispiel dazu angewiesen werden, eine Wartungs-Seite zu schalten, bevor die zugrundeliegenden Systemdienste heruntergefahren werden. Dieser Ansatz hilft außerdem, dem Nutzer die Verantwortlichkeit hinter der Ausführung von *Playbook*s deutlicher zu machen.

Literaturverzeichnis

[1] Ansible project contributors: *Ansible architecture — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html#modules. – [Online; accessed 31-January-2022]

[2] Ansible project contributors: *Ansible concepts — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/user_guide/basic_concepts.html. – [Online; accessed 31-January-2022]

[3] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html#term-YAML. – [Online; accessed 31-January-2022]

[4] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html#term-Modules. – [Online; accessed 31-January-2022]

[5] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html#term-Push-Mode. – [Online; accessed 31-January-2022]

[6] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html#term-Pull-Mode. – [Online; accessed 31-January-2022]

[7] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_

appendices/glossary.html#term-Host-Specifier. – [Online; accessed 31-January-2022]

[8] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html#term-Task. – [Online; accessed 31-January-2022]

[9] Ansible project contributors: *Glossary — Ansible Documentation*. 2021. – URL https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html#term-Plays. – [Online; accessed 31-January-2022]

[10] Haoues, Mariem ; Sellami, Asma ; Ben-Abdallah, Hanêne ; Cheikhi, Laila: A guideline for software architecture selection based on ISO 25010 quality related characteristics. In: *International Journal of System Assurance Engineering and Management* 8 (2017), Nr. 2, S. 886–909

[11] Progress Software Corporation: *About Chef Workstation*. 2021. – URL https://docs.chef.io/workstation/. – [Online; accessed 31-January-2022]

[12] Progress Software Corporation: *About Cookbooks*. 2021. – URL https://docs.chef.io/cookbooks/. – [Online; accessed 31-January-2022]

[13] Progress Software Corporation: *About Nodes*. 2021. – URL https://docs.chef.io/nodes/. – [Online; accessed 31-January-2022]

[14] Progress Software Corporation: *About Recipes*. 2021. – URL https://docs.chef.io/recipes/. – [Online; accessed 31-January-2022]

[15] Progress Software Corporation: *About Resources*. 2021. – URL https://docs.chef.io/resource/. – [Online; accessed 31-January-2022]

[16] Progress Software Corporation: *About Roles*. 2021. – URL https://docs.chef.io/roles/. – [Online; accessed 31-January-2022]

[17] Progress Software Corporation: *About Run-lists*. 2021. – URL https://docs.chef.io/run_lists/. – [Online; accessed 31-January-2022]

[18] Progress Software Corporation: *Chef Infra Client Overview*. 2021. – URL https://docs.chef.io/chef_client_overview/. – [Online; accessed 31-January-2022]

[19] Progress Software Corporation: *Chef Infra Overview*. 2021. – URL https://docs.chef.io/chef_overview/. – [Online; accessed 31-January-2022]

[20] Progress Software Corporation: *Chef Infra Server Overview*. 2021. – URL https://docs.chef.io/server/. – [Online; accessed 31-January-2022]

[21] Progress Software Corporation: *Chef and Ansible*. 2022. – URL https://www.chef.io/ansible. – [Online; accessed 31-January-2022]

[22] Progress Software Corporation: *Chef Software DevOps Automation Solutions | Chef*. 2022. – URL https://www.chef.io/. – [Online; accessed 31-January-2022]

[23] R. Barnes, J. Hoffman-Andrews, D. McCarney, J. Kasten: *RFC 8555 - Automatic Certificate Management Environment (ACME)*. 2019. – URL https://datatracker.ietf.org/doc/html/rfc8555. – [Online; accessed 31-January-2022]

[24] Red Hat, Inc.: *The AWX project*. 2020. – URL https://www.ansible.com/products/awx-project/faq. – [Online; accessed 31-January-2022]

[25] Red Hat, Inc.: *Ansible is Simple IT Automation*. 2022. – URL https://www.ansible.com/. – [Online; accessed 31-January-2022]

[26] Red Hat, Inc.: *Red Hat Ansible - Automation controller*. 2022. – URL https://www.ansible.com/products/controller. – [Online; accessed 31-January-2022]

[27] T. Ylonen, C. Lonvick: *RFC 4251 - The Secure Shell (SSH) Protocol Architecture*. 2006. – URL https://datatracker.ietf.org/doc/html/rfc4251. – [Online; accessed 31-January-2022]

[28] Van Steen, Maarten ; Tanenbaum, Andrew S.: *Distributed systems*. Maarten van Steen Leiden, The Netherlands, 2017

Glossar

Ansible Automatisierungsplattform von *Red Hat, Inc.* [25] - Im Rahmen dieser Arbeit synonym für *AWX*.

Ansible Automation Controller Web-Interface zur Steuerung von *Ansible*, ehemals bekannt als *Ansible Tower* [26].

Ansible Tower Frühere Bezeichnung von *Ansible Automation Controller* [26].

AWX Web-Interface zur Steuerung von *Ansible*. Open Source Upstream Projekt für *Ansible Automation Controller* [24].

Chef Automatisierungs-Ökosystem von *Progress Software Corporation* [22] - Im Rahmen dieser Arbeit synonym für *Chef Infra*.

Chef Automate Chef Terminologie - Dashboard für verschiedene Chef-Produkte, unter anderem *Chef Infra*.

Chef Infra Chef Terminologie - Automatisierungs-Plattform [19].

Chef Infra Client Chef Terminologie - Software, welche regelmäßig die *Run-List* für die ausführende *Node* vom *Chef Server* bezieht und die darin enthaltenen *Recipe*s abarbeitet [18].

Chef Infra Server Chef Terminologie - Software, auf welcher Konfigurationen von *Chef Workstation*s abgelegt und von *Chef Infra Client*s abgerufen werden können [20].

Chef Server Chef Terminologie - System auf welchem *Chef Infra* installiert ist [19].

Chef Workstation Chef Terminologie - System, auf welchem die gleichnamige Software zur Verwaltung eines *Chef Server*s installiert ist [11].

Control Node Ansible Terminologie - System, auf welchem Ansible installiert ist und durch welches *Managed Node*s gesteuert werden [2].

Cookbook Chef Terminologie - Sammlung von *Recipe*s mit Metadaten [12]. Vergleichbar mit *Project*.

Host Ansible Terminologie - Alternative Bezeichnung für *Managed Node* [2].

Host Specifier Ansible Terminologie - Erlaubt es, eine Submenge des verwendeten *Inventory*s anhand von *Group*- oder *Host*-Namen auszuwählen [7].

Inventory Ansible Terminologie - Liste von *Managed Node*s. Dient als Ziel für die Ausführung eines *Playbook*s [2].

Managed Node Ansible Terminologie - System, welches durch eine *Control Node* gesteuert wird. Alternativ auch als *Host* bezeichnet. Teil eines *Inventory*s [2].

Module Ansible Terminologie - Code-Einheit, welche von Ansible auf *Managed Node*s ausgeführt wird [2]. Kann in verschiedenen Sprachen geschrieben werden, es wird jedoch empfohlen, *Python* zu nutzen [4]. Vergleichbar mit *Resource*.

Module Argument Ansible Terminologie - Parameter für die Ausführung eines *Module*s. Werden durch den ausführenden *Task* gesetzt [8].

Node Chef Terminologie - System, auf welchem *Chef Infra Client* installiert ist und seine *Run-List* von einem *Chef Server* bezieht [13].

Play Ansible Terminologie - *Host Specifier* zusammen mit geordneter Liste von *Task*s. Teil eines *Playbook*s [9]. Vergleichbar mit *Recipe*.

Playbook Ansible Terminologie - Geordnete Liste von *Play*s. Beschreibt die auf der *Managed Node* auszuführenden Aufgaben. Definiert in *YAML* [2]. Vergleichbar mit *Run-List*.

Progress Software Corporation Unternehmen, welches *Chef* anbietet.

Project Ansible Terminologie - Sammlung von *Playbook*s. Vergleichbar mit *Cookbook*.

Pull Mode Betriebsart, bei welcher ein Client in regelmäßigen Abständen eine Verbindung zu einem Server aufbaut um ausstehende Konfiguartionen abzufragen. Gegenstück zu *Push Mode*. Betriebsart von *Chef* [19]. Optionale Betriebsart von *Ansible* [6].

Push Mode Betriebsart, bei welcher ein Server eine Verbindung zu einem Client aufbaut um Konfigurationen zu übertragen. Gegenstück zu *Pull Mode*. Standardmäßige Betriebsart von *Ansible* [5].

Python Objektorientierte Interpreter-Sprache, mit welcher *Module*s verfasst werden. Vergleichbar mit *Ruby*.

Recipe Chef Terminologie - *Ruby*-Datei, welche den erwünschten Konfigurations-Stand beschreibt. Teil eines *Cookbook*s. Kann Teil von anderen Recipes sein [14]. .

Red Hat, Inc. Unternehmen, welches *Ansible* anbietet.

Resource Chef Terminologie - Hilfs-Code, mit dessen Hilfe in *Recipe*s der erwünschte Konfigurations-Status beschrieben wird [15]. Vergleichbar mit *Module*s.

Role Chef Terminologie - *Run-List* mit Metadaten. Erlaubt einer *Run-List* somit, dass sie sich aus anderen *Run-List*s zusammensetzt [16].

Ruby Objektorientierte Interpreter-Sprache, mit welcher *Recipe*s verfasst werden. Vergleichbar mit *Python*.

Run-List Chef Terminologie - Geordnete Liste von *Recipe*s und *Role*s [17]. Vergleichbar mit *Playbook*.

SSH Protokoll für den Aufbau von gesicherten Verbindungen in unsicheren Netzwerken [27].

Task Ansible Terminologie - Auszuführende Aktion, welche auf ein *Module* verweist und *Module Argument*s für dieses liefert. Teil eines *Play*s [2].

YAML Menschenlesbare Auszeichnungssprache mit der Dateiendung *.yaml* bzw *.yml*. Superset von JSON, welches auf Einrückung an Stelle von Klammerung zurückgreift [3].

yml Alternative Dateiendung für *YAML*-Dateien.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original