Denis Racz

Einstieg in Ansible

Schritt für Schritt vom ersten Playbook zur professionellen Automatisierung

Mit zahlreichen Praxisbeispielen und Best Practices

mitp

Inhaltsverzeichnis

Kapitel 1

Die Sprache der Infrastruktur: Willkommen bei Ansible

1.1 Warum man Infrastruktur heute nicht mehr »per Hand« verwaltet

Stellen Sie sich vor, Sie müssten in einem Unternehmen mit zehn, zwanzig oder gar hundert Servern jeden einzelnen von Hand einrichten: Benutzerkonten anlegen, Software installieren, Konfigurationsdateien anpassen, Systemupdates einspielen. Jede noch so kleine Änderung müssten Sie manuell auf jedem System wiederholen. Fehler wären vorprogrammiert – sei es durch Flüchtigkeit, durch missverständliche Anleitungen oder durch minimale Unterschiede zwischen den Systemen.

Früher war das tatsächlich oft Realität. Traditionell wurden IT-Systeme manuell konfiguriert, doch in einer Zeit, in der IT-Systeme immer komplexer werden, Cloud-Plattformen wie AWS, Azure oder Google Cloud Platform dominieren und Unternehmen auf schnelle Reaktionsfähigkeit angewiesen sind, ist dieses Vorgehen schlicht nicht mehr zeitgemäß.

Genau hier setzt Infrastructure as Code (IaC) an: Anstatt manuell Server und Systeme zu konfigurieren, wird Infrastruktur wie ein Softwareprojekt behandelt – in Code gegossen, versioniert, überprüft und automatisiert ausgeführt.

Dies bringt zahlreiche Vorteile mit sich:

- **Konsistenz:** Jede Umgebung (z.B. Staging-, Test- und Produktionsumgebungen) kann identisch eingerichtet werden.
- **Wiederholbarkeit:** Änderungen und Deployments können automatisiert und reproduzierbar durchgeführt werden.
- **Effizienz:** Automatisierung spart Zeit und reduziert menschliche Fehler.
- **Skalierbarkeit:** Infrastruktur kann schnell und flexibel an neue Anforderungen angepasst werden.

Infrastructure as Code bedeutet, dass Sie Ihre IT-Infrastruktur – also z. B. Server, Netzwerke, Datenbanken oder Firewalls – mithilfe von Textdateien beschreiben, die ein Programm später automatisch ausführt. Diese Beschreibungen werden oft in einer menschenlesbaren Sprache verfasst, beispielsweise YAML (Yet Another Markup Language), wie es Ansible nutzt.

Analogie

So wie ein Architekt einen Bauplan erstellt, der festlegt, wie ein Haus gebaut werden soll, schreiben Sie als IT-Administrator einen »Bauplan« für Ihre Infrastruktur. Und Ansible ist dann die Baufirma, die diesen Plan automatisch umsetzt – immer gleich, immer zuverlässig.

1.1.1 Traditionelle IT vs. Infrastructure as Code

Merkmal	Traditionelle IT-Verwaltung	Infrastructure as Code
Provisionierung	Manuelle Einrichtung	Automatisiert per Code
Konfiguration	Manuelle Änderung	Deklarative Playbooks
Reproduzierbarkeit	Schwer nachvollziehbar	Versionierte Konfiguration
Fehlersuche	Zeitaufwendig, inkonsistent	Strukturierte, dokumentierte Prozesse

Tabelle 1.1: Vergleich von tradioneller IT und Infrastructure as Code

Die Umstellung auf Infrastructure as Code bringt zahlreiche Vorteile mit sich:

- **Automatisierung:** Aufgaben, die früher manuell erledigt wurden, laufen jetzt automatisiert ab.
- **Konsistenz:** Alle Systeme werden identisch konfiguriert – Konfigurationsdrift wird vermieden.
- **Versionierbarkeit:** Jede Änderung am Infrastruktur-Code kann nachvollzogen, dokumentiert und bei Bedarf rückgängig gemacht werden.
- **Reproduzierbarkeit:** Ein neuer Server kann auf Knopfdruck genauso eingerichtet werden wie ein bestehender.
- **Teamarbeit:** Der Code kann gemeinsam entwickelt, geprüft und getestet werden – ganz wie bei der Softwareentwicklung.

Diese Eigenschaften machen IaC nicht nur in großen IT-Abteilungen interessant, sondern auch für kleine Teams, Start-ups oder Einzelpersonen, die ihre Arbeitsweise professionalisieren wollen.

1.1.2 Deklarativ statt prozedural: Denken in Zielzuständen

Ein zentrales Konzept, das Ansible von vielen traditionellen Ansätzen zur Systemverwaltung unterscheidet, ist die sog. *deklarative Arbeitsweise*. Statt in Anweisungen (»Tu dies, dann das«) denken Sie in Zielzuständen: Sie beschreiben, *wie* ein System am Ende aussehen soll, und Ansible kümmert sich darum, die nötigen Schritte dorthin zu finden und umzusetzen.

Traditionelle (imperative) Verwaltung

In klassischen Skripten – etwa mit Bash, PowerShell oder Python – schreiben Sie Schritt für Schritt vor, *wie* ein gewünschtes Ergebnis zu erreichen ist:

```
#!/bin/bash

# Benutzer anlegen
useradd erika

# Home-Verzeichnis anlegen
mkdir -p /home/erika/.ssh
chown -R erika:erika /home/erika
chmod 700 /home/erika/.ssh

# SSH-Key setzen
echo "ssh-rsa AAAAB3..." > /home/erika/.ssh/authorized_keys
chown erika:erika /home/erika/.ssh/authorized_keys
chmod 600 /home/erika/.ssh/authorized_keys
```

Der Administrator legt dabei exakt fest, welche Schritte in welcher Reihenfolge ausgeführt werden. Das zentrale Problem solcher Skripte ist jedoch, dass sie nicht zustandsbasiert arbeiten. Sie prüfen in der Regel nicht, ob ein gewünschter Zustand bereits erreicht ist, und lassen sich daher nur eingeschränkt wiederverwenden oder gefahrlos mehrfach ausführen. Existiert ein Benutzer bereits oder ist ein SSH-Schlüssel schon hinterlegt, schlägt das Skript häufig fehl – oder liefert in leicht veränderten Umgebungen plötzlich andere Ergebnisse.

Der deklarative Ansatz von Ansible

Mit Ansible definieren Sie, **was** gelten soll, nicht, **wie** man dorthin kommt. Zum Beispiel:

```
- name: Benutzer anlegen
  ansible.builtin.user:
    name: erika
    state: present

- name: SSH-Key setzen
  ansible.posix.authorized_key:
    user: erika
    key: "ssh-rsa AAAAB3..."
```

Sie müssen dabei nicht mehr jeden einzelnen Shell-Befehl kennen oder darauf achten, ob ein Benutzer bereits existiert, ob das SSH-Verzeichnis korrekt angelegt ist oder ob die Dateiberechtigungen stimmen. Das jeweilige Ansible-Modul kapselt diese Details und stellt sicher, dass der gewünschte Zustand zuverlässig, wiederholbar und nachvollziehbar erreicht wird.

Nur wenn der gewünschte Zustand noch nicht erreicht ist, greift Ansible ein. Dieses Prinzip nennt man Idempotenz: Die gleiche Automatisierung kann mehrfach ausgeführt werden, ohne Systeme jedes Mal erneut zu verändern oder »kaputt zu konfigurieren«.

Die deklarative Arbeitsweise mag anfangs etwas ungewohnt wirken, doch sie bildet die Grundlage für stabile, überprüfbare und automatisierte IT-Infrastruktur. Sie schreiben nicht mehr Code, um Aufgaben zu erledigen, sondern beschreiben den gewünschten Zustand, und Ansible sorgt für die Umsetzung.

1.1.3 Warum der deklarative Ansatz insbesondere Einsteigern hilft

Der deklarative Ansatz verändert die Lernkurve für Einsteiger grundlegend. Statt komplexe Befehlsfolgen und deren Seiteneffekte im Detail verstehen zu müssen, beschreiben Sie mit Ansible den gewünschten Zielzustand eines Systems. Wie dieser Zustand erreicht wird, übernimmt Ansible.

Früher war es notwendig, tiefgreifende technische Details zu kennen, um etwa ein RAID korrekt einzurichten oder Benutzer konsistent anzulegen. Mit Ansible lassen sich dieselben Ergebnisse heute mit wenigen, gut lesbaren YAML-Zeilen beschreiben. Das senkt die Einstiegshürde deutlich:

- **Komplexe Aufgaben werden verständlich** – auch ohne jahrelange Erfahrung.
- **Anleitungen zur Automatisierung sind gut lesbar** – man erkennt sofort, welches Ziel damit erreicht werden soll.
- **Fehleranfällige manuelle Schritte entfallen** – Ansible sorgt für Wiederholbarkeit und Korrektheit.
- **Einsteiger werden schneller produktiv** – viele Aufgaben lassen sich schon nach wenigen Wochen sicher automatisieren.

Zusätzlich unterstützt Ansible diesen Ansatz durch bewusst einfache Rahmenbedingungen. Ansible arbeitet agentlos (engl. *agentless*): Auf den Zielsystemen muss keine zusätzliche Software installiert werden, abgesehen von einer vorhandenen Python-Laufzeitumgebung, die in den meisten Linux-Distributionen bereits mitgeliefert wird. Es läuft auf gängigen Plattformen wie Linux, macOS oder Windows (via WSL) und kommt mit einer überschaubaren Installation aus. Die verwendete YAML-Syntax ist bewusst einfach gehalten und erleichtert das Lesen und Verstehen auch ohne tiefgehende Programmierkenntnisse.

Mit Ansible denken Sie daher nicht mehr in einzelnen Befehlen, sondern in gewünschten Zuständen. Das vereinfacht nicht nur den Alltag erfahrener Admi-

nistratoren, sondern ermöglicht auch Einsteigern, früh produktive und nachvollziehbare Automatisierungen umzusetzen.

1.2 Was ist Ansible?

Ansible ist eines der führenden Werkzeuge für Infrastructure as Code und zeichnet sich durch seine einfache Handhabung und hohe Flexibilität aus. Es wurde 2012 von Michael DeHaan entwickelt und hat sich seitdem zu einem Standard für IT-Automatisierung entwickelt.

1.2.1 Wie Ansible zu seinem Namen kam

Der Name Ansible stammt – wie so viele Begriffe in der Tech-Welt – aus der Science-Fiction. Genauer ausgedrückt beschreibt ein Ansible ein fiktives Gerät, das in der Lage ist, Informationen augenblicklich über beliebige Entfernungen hinweg zu übertragen – also schneller als Lichtgeschwindigkeit. Die Idee dieses überlichtschnellen Kommunikationsmittels wurde erstmals von der amerikanischen Autorin Ursula K. LeGuin in ihrem Roman »Raconnas Welt« (1966) eingeführt. In ihrem Werk diente das Ansible als Verbindung zwischen weit entfernten Welten.

Später wurde der Begriff von anderen Science-Fiction-Autoren weiterverwendet, darunter auch Orson Scott Card in »Das große Spiel« (1977). In diesem Roman spielt das Ansible eine zentrale Rolle in der militärischen Fernsteuerung ganzer Raumschiff-Flotten – ohne Zeitverzögerung, zuverlässig und synchron.

Diese Metapher griff Michael DeHaan, einer der ursprünglichen Entwickler von Ansible, auf. Für ihn war der Name eine perfekte Beschreibung dessen, was seine Software leisten sollte: eine zentrale Instanz, die viele entfernte Systeme steuern kann – sofort, koordiniert und ohne unnötigen Ballast. Statt Befehle manuell auf jedem Server auszuführen, übernimmt Ansible diese Aufgabe automatisch, als stünde es »in direkter Verbindung« mit allen Maschinen – ganz wie das fiktive Original. So wurde aus einer literarischen Vision eine moderne Automatisierungslösung – und aus dem Begriff *Ansible* ein Synonym für effiziente IT-Steuerung über jede Distanz hinweg.

1.2.2 Die Stärken von Ansible

Ansible hat sich in den letzten Jahren als eines der am häufigsten eingesetzten Werkzeuge für IT-Automatisierung etabliert. Der Grund dafür liegt weniger in einzelnen Spezialfunktionen als in einem insgesamt sehr pragmatischen Ansatz: Ansible ist leicht zu erlernen, arbeitet agentenlos und beschreibt Infrastrukturzustände klar und deklarativ.

Statt komplexer Programmlogik formulieren Sie in übersichtlichen YAML-Dateien, wie ein System aussehen soll. Ansible übernimmt anschließend die Umsetzung dieses Zielzustands. Dadurch lassen sich wiederkehrende Aufgaben automatisieren, menschliche Fehler reduzieren und konsistente, reproduzierbare Umgebungen schaffen – vom einzelnen Server bis hin zu größeren, mehrstufigen Architekturen.

Gegenüber anderen Automatisierungstools bietet Ansible mehrere Vorteile :

- **Agentenloses Arbeiten:** Auf den Zielsystemen ist keine dauerhaft laufende Software erforderlich. Ansible nutzt bestehende Mechanismen wie SSH, WinRM oder APIs, was Betrieb und Absicherung vereinfacht.
- **Einfache und gut lesbare Syntax:** Automatisierungen werden in YAML beschrieben und sind auch ohne Programmierhintergrund gut nachvollziehbar.
- **Deklarativer Ansatz und Idempotenz:** Aufgaben werden nur dann ausgeführt, wenn sie tatsächlich notwendig sind. Wiederholte Ausführungen führen nicht zu unerwünschten Nebeneffekten.
- **Modularität und Wiederverwendbarkeit:** Durch Roles, Templates und Variablen lassen sich Automatisierungen sauber strukturieren und projektübergreifend einsetzen.
- **Breite Unterstützung:** Ansible kann Linux-, macOS- und Windows-Systeme ebenso verwalten wie Netzwerkgeräte, Virtualisierungsplattformen, Container-Umgebungen und Cloud-Dienste.
- **Kostenlos**: Es ist frei verfügbar und wird aktiv weiterentwickelt.

Abb. 1.1: Einige Systeme und Plattformen, mit denen Ansible interagieren kann

Anders als bei komplexeren Tools wie Terraform oder Puppet können Sie mit Ansible oft schon nach wenigen Stunden erste produktive Ergebnisse erzielen. Deshalb eignet es sich hervorragend für den Einstieg in die Welt der IT-Automatisierung.

Praxistipp

Ansible eignet sich besonders gut für Aufgaben wie das Installieren von Softwarepaketen, das Anpassen von Konfigurationsdateien oder das Starten von Diensten – also für das, was Systemadministratoren tagtäglich tun.

1.2.3 Einsatzbereiche in der Praxis

Ansible ist ein vielseitiges Werkzeug zur Automatisierung von IT-Infrastruktur. Es eignet sich besonders für Aufgaben, die wiederholt, zuverlässig und auf mehreren Systemen gleichzeitig ausgeführt werden müssen.

Typische Einsatzbereiche sind:

- Konfigurationsmanagement: Systeme in einen definierten Zustand versetzen, z. B. Pakete installieren oder Dienste aktivieren
- Provisionierung: Neue Server oder Cloud-Ressourcen automatisiert bereitstellen
- Deployment: Anwendungen inklusive Abhängigkeiten auf mehreren Hosts ausrollen
- Orchestrierung: Abhängige Systeme in der richtigen Reihenfolge einrichten oder steuern (z. B. erst Datenbank, dann Webserver)
- Patch-Management: Sicherheitsupdates zeitgleich auf vielen Systemen verteilen

Kurz: Ansible ist dann besonders hilfreich, wenn viele Systeme effizient, wiederholbar und nachvollziehbar verwaltet werden sollen, egal ob im Rechenzentrum, in der Cloud oder im gemischten Betrieb.

1.2.4 Ist Ansible mächtig genug?

Viele IT-Administratoren, die seit Jahren Shell- oder Bash-Skripte zur Systemautomatisierung einsetzen, stehen Ansible zunächst mit einer gewissen Skepsis gegenüber. Sie fragen sich, ob ein deklaratives Tool wie Ansible tatsächlich alle komplexen Anforderungen abdecken kann oder ob es nicht doch sinnvoller ist, bei ihren bewährten, individuell geschriebenen Skripten zu bleiben.

Diese Sorge ist verständlich, aber unbegründet. Tatsächlich bietet Ansible nicht nur eine alternative Methode zur Automatisierung, sondern eröffnet Möglichkei-

ten, die weit über klassische Shell-Skripte hinausgehen – insbesondere was Wartbarkeit, Wiederverwendbarkeit, Lesbarkeit und Flexibilität betrifft.

Skripte ersetzen – ohne Einbußen bei der Kontrolle

Ein zentrales Missverständnis ist die Annahme, Ansible sei »zu einfach« oder »zu starr«, um komplexe Szenarien zu bewältigen. Dabei kann Ansible:

- komplexe Abläufe dynamisch steuern,
- Daten aus vorherigen Tasks wiederverwenden,
- abhängig vom Systemzustand verschiedene Aktionen ausführen,
- Schleifen, Bedingungen und Variablenkontexte nutzen
- und sogar direkt Shell-Befehle einbetten, wenn nötig.

All das in einer klar strukturierten, modularen und wiederverwendbaren Form.

Beispiel: Passwortwechsel während eines Datenbank-Setups

Ein häufiger Zweifel bei deklarativen Werkzeugen ist die Frage, ob sie auch mit Zustandswechseln innerhalb eines Ablaufs umgehen können. Genau das ist in der Praxis jedoch oft notwendig, etwa bei der initialen Einrichtung einer Datenbank.

Ein typisches Beispiel: Sie richten eine neue Datenbankinstanz ein, die zunächst mit einem bekannten Default-Passwort wie »changeme« erreichbar ist. Ziel der Automatisierung ist es

1. sich mit dem Default-Passwort zu verbinden,
2. ein neues Passwort zu setzen,
3. und anschließend direkt mit den neuen Zugangsdaten weiterzuarbeiten, ohne den Ablauf zu unterbrechen oder ein zweites Mal starten zu müssen.

Mit Bash-, PowerShell- oder Python-Skripten ist das häufig mit temporären Hilfsdateien, Zwischenspeicherung oder komplexen Kontrollstrukturen verbunden. In Ansible hingegen lässt sich dieses Szenario elegant in einem einzigen Playbook abbilden.

Ansible hingegen arbeitet kontextbasiert weiter, ohne dass ein zweites Skript gestartet werden oder manuell eingegriffen werden muss. Es nutzt zunächst das alte Passwort, wechselt dann intern zu den neuen Zugangsdaten und setzt den Ablauf nahtlos fort.

Warum das mehr ist als nur ein »Ersatz« für Bash-Skripte

Das vorherige Beispiel zeigt bewusst mehr als nur eine technische Spielerei. Es verdeutlicht, dass Ansible nicht lediglich einzelne Befehle automatisiert, sondern komplexe Abläufe strukturiert abbilden kann – inklusive Zustandswechseln,

Abhängigkeiten und Wiederverwendbarkeit. Genau an dieser Stelle unterscheidet sich Ansible grundlegend von klassischen Skriptlösungen.

Zusätzlich zu solchen dynamischen Kontrollflüssen bietet Ansible:

- **Templating mit Jinja2:** Konfigurationsdateien lassen sich mit Platzhaltern und einfacher Logik generieren.
- **Roles und Wiederverwendung:** Aufgaben können modular aufgebaut, standardisiert und projektübergreifend verwendet werden.
- **Variablen in mehreren Ebenen:** Werte lassen sich Host-, Gruppen- oder kontextabhängig sowie zur Laufzeit setzen.
- Handler **und Idempotenz:** Änderungen werden nur dann durchgeführt, wenn sie tatsächlich notwendig sind – ideal für wiederholte Ausführungen.
- **Vault für sichere Passwörter:** Zugangsdaten und sensible Daten können verschlüsselt verwaltet werden.

Ansible ist damit kein reiner Ersatz für Bash-Skripte, sondern ein vollständiges Automatisierungs-Framework für produktive Umgebungen. Es nimmt Administratoren keine Kontrolle ab, sondern ersetzt implizite Logik durch nachvollziehbare Struktur, Wiederverwendbarkeit und Sicherheit.

Sie müssen sich daher nicht zwischen Skripten und Ansible entscheiden. Vielmehr erlaubt Ansible, bestehende Automatisierungen schrittweise zu verbessern und gleichzeitig verständlicher und wartbarer zu gestalten – für Sie selbst ebenso wie für Ihre Kolleginnen und Kollegen.

1.3 Einführung in Ansible

1.3.1 Die wichtigsten Komponenten

Bevor Sie mit Ansible produktiv arbeiten können, ist es wichtig, die grundlegenden Bausteine zu verstehen, aus denen sich Ihre Automatisierung zusammensetzt. Ansible verfolgt dabei einen modularen und klar strukturierten Ansatz. Vier zentrale Konzepte bilden das Fundament: Playbooks, Module, Inventories und Roles. Sie greifen wie Zahnräder ineinander und ermöglichen es, selbst komplexe Aufgaben systematisch und wiederverwendbar zu automatisieren.

Im Mittelpunkt stehen die Playbooks. Dabei handelt es sich um einfache Textdateien im YAML-Format, in denen beschrieben wird, welche Aufgaben auf welchen Systemen ausgeführt werden sollen. Ein Playbook ist gewissermaßen das »Drehbuch« Ihrer Automatisierung – es legt fest, welche Schritte wann und in welcher Reihenfolge erfolgen.

Damit diese Aufgaben nicht jedes Mal im Detail programmiert werden müssen, greift Ansible auf eine große Sammlung von Modulen zurück. Diese Module sind

kleine, spezialisierte Programme, die bestimmte Aufgaben übernehmen – zum Beispiel das Anlegen von Benutzerkonten, das Installieren von Software oder das Starten eines Dienstes. Sie sind der eigentliche »Motor« unter der Haube und ersparen Ihnen viel Aufwand.

Ein weiterer zentraler Bestandteil ist das sogenannte Inventory. Hierbei handelt es sich um eine Datei oder Datenquelle, in der festgelegt ist, auf welchen Servern Ansible die beschriebenen Aufgaben ausführen soll. Diese Server können lokal, in einer Cloud oder in einer hybriden Umgebung liegen. Durch die klare Trennung zwischen *Was* (Playbook) und *Wo* (Inventory) bleibt Ihre Automatisierung übersichtlich und flexibel.

Für größere oder wiederkehrende Aufgaben empfiehlt es sich, mit Roles zu arbeiten. Roles bündeln alle relevanten Bestandteile – also Aufgaben, Variablen, Vorlagen und Dateien – in einer einheitlichen Struktur. Dadurch lassen sich einzelne Automatisierungskomponenten leicht wiederverwenden, teilen oder modular kombinieren. Wenn Sie zum Beispiel eine Role für das Einrichten eines Webservers erstellt haben, können Sie diese in verschiedenen Projekten oder Umgebungen immer wieder einsetzen, ohne von vorn beginnen zu müssen.

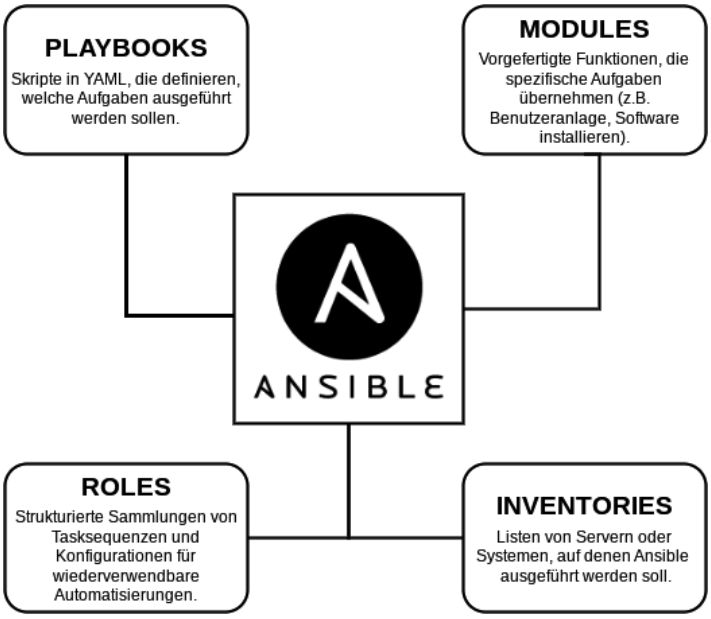

Abb. 1.2: Die Komponenten von Ansible

Zusammen ermöglichen Ihnen diese vier Komponenten ein sauberes, nachvollziehbares und skalierbares Arbeiten mit Ansible – ganz gleich, ob Sie eine Handvoll Testsysteme oder eine komplexe Serverlandschaft verwalten möchten.

Stichwortverzeichnis