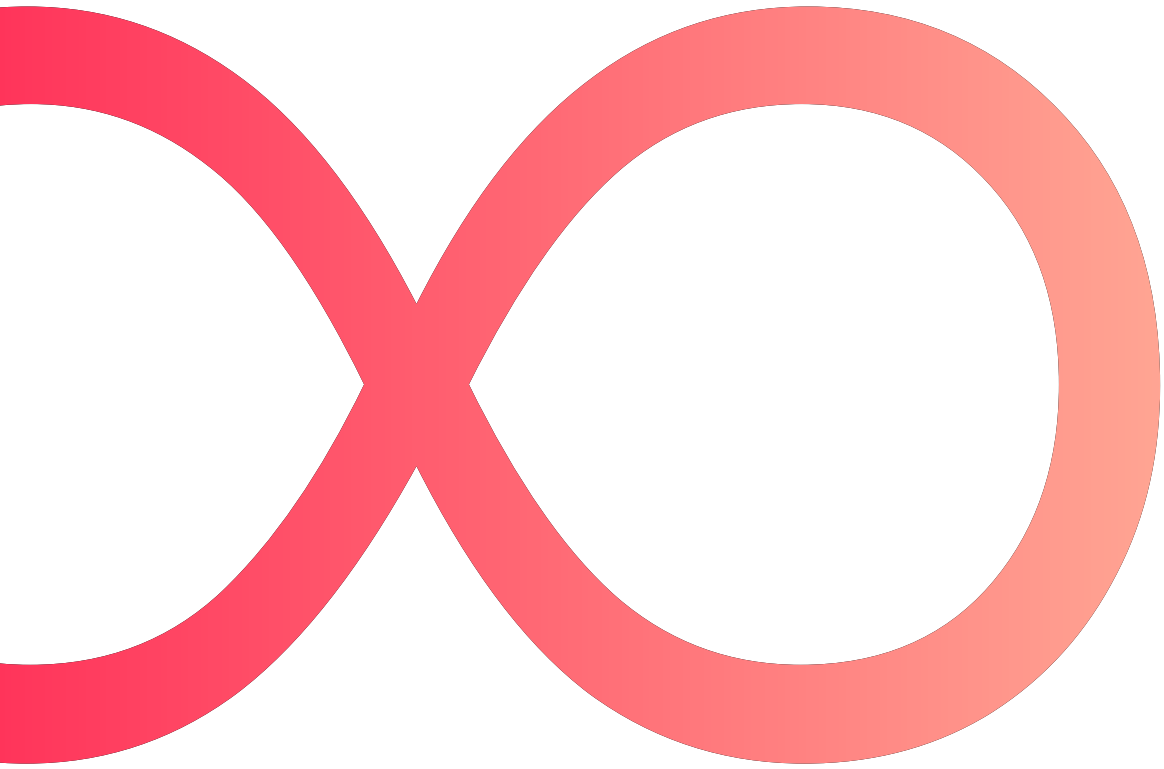
makandra



# INFRASTRUCTURE AS CODE (IAC)

Eine Einführung

makandra

# INFRASTRUCTURE AS CODE (IAC)

## Eine Einführung

Wenn Sie sich dieses Whitepaper herunterladen, dann haben Sie aller Wahrscheinlichkeit nach mit dem Aufbau von guter Infrastruktur für Ihre Anwendungen zu tun. Oder Sie planen, zukünftig damit zu tun zu haben und wollen sich deshalb vorher über die Best Practices informieren. In jedem Fall werden wir nachfolgend eine kurze Zusammenfassung zu einem zentralen Thema in der modernen DevOps-Welt geben: Infrastructure as Code (IaC). Wir zeigen, weshalb es Sinn macht, sich intensiv damit auseinanderzusetzen und warum IaC ein Standard im professionellen Betrieb von Webanwendungen sein sollte.

makandra

# INHALTSVERZEICHNIS

# Vor langer, langer Zeit...

... da hörten wir noch Kassetten und tranken Bluna.
Nein, im Ernst. Solange sind die Zeiten noch nicht her, in denen wir physische Server und Infrastruktur im Keller stehen hatten und jede Anwendung auf einer eigenen dedizierten Hardware betrieben haben. Wollte man die Infrastruktur ausbauen, musste man neue Hardware ordern. Da konnte schon mal die ein oder andere Woche oder gar Monate ins Land gehen, bevor man neue Server bereitstellen konnte.
Und auch wenn Systemadministratoren bereits seit den 1990er-Jahren Skripte zur Verwaltung ihrer Infrastruktur verwendeten, **beginnt die Geschichte und Bedeutung von Infrastructure as Code (IaC) mit der Virtualisierung von Servern.**
Diese lösten das zentrale Problem der bisherigen physischen Server: schlechte Ressourcennutzung und geringe Flexibilität. Von nun an konnte man mehrere virtuelle Maschinen (VMs) auf einem einzigen physischen Server betreiben. Mit dem **Einzug der großen Cloud-Anbieter in den 2000er-Jahren** beschleunigte sich diese Entwicklung noch. Es war nun möglich, neue VMs in wenigen Minuten hochzufahren und praktisch eine beliebige Anzahl von Instanzen zu betreiben, ohne die dafür notwendige Hardware zu besitzen.
Schnell brachte dies ein neues Problem mit sich. Mit der neu gewonnenen Flexibilität, virtuelle Maschinen beliebig starten zu können, stieg die Anzahl an Servern schnell an. Es entstand die Herausforderung, diese ständig wachsende und sich verändernde Serverlandschaft aktuell zu halten und Konfigurationsabweichungen zu vermeiden.
Das genaue Wissen über das Set-up der Infrastruktur lag dabei oft in den Händen einiger weniger Personen. Kamen neue Mitarbeitende dazu oder verließen zentrale Personen das Unternehmen, stellte das eine große Herausforderung dar. Es passierten Fehler, weil Änderungen nicht dokumentiert oder weitergegeben wurden. Erstellte man langwierige Dokumentationen, waren diese meist schon wieder veraltet, bevor sie überhaupt herausgegeben wurden.

Kurzum: Es wurde schnell klar, dass ein automatisierter und für jeden nachvollziehbarer Ansatz benötigt wurde. Hierfür entstanden Ende der 2000er neue Werkzeuge wie Puppet und Chef, die es erlaubten, die **Konfiguration der Infrastruktur durch Code zu definieren**.

### Exkurs – Zusammenklicken einer Infrastruktur in AWS und warum das nicht so einfach ist

Beispiel: Wir betrachten eine kleine Webanwendung, für die man sich mit AWS eine Infrastruktur aufsetzt.

Für die Benutzerschnittstelle benötigt man einen Web-Server, der den statischen Content wie HTML, CSS und JavaScript über ein S3 Bucket bereitstellt.
Außerdem hat man einen Anwendungsserver im Einsatz, der Benutzeranfragen verarbeitet und Antworten generiert, indem er mit der Datenbank kommuniziert. Dieser wird auf einer EC2-Instanz bereitgestellt. Dann hat man noch einen Datenbankserver, der die Daten- oder Back-End-Schicht der Anwendung darstellt und – zum Beispiel – auf AWS RDS läuft.

Jede dieser Instanzen befindet sich in einer Auto-Scaling-Gruppe mit einem Load-Balancer davor (mit Ausnahme der Datenbank). Zusätzlich will man in der Regel auch noch weitere Infrastrukturen wie Caches, Queues, Firewall-Regeln, IAM oder SSL-Zertifikate definieren.

Wenn man diese Infrastruktur über die AWS-Konsole erstellen möchte, muss man manuell durch verschiedene Bildschirme navigieren, um die Infrastruktur hochzufahren. Das ist komplex, fehleranfällig und die Historie der Änderungen sind schwer nachvollziehbar.

# IaC - Definition und Vorteile

Wie der Name schon sagt, versteht man unter Infrastructure as Code einen Ansatz zur Automatisierung der Infrastruktur, der auf Praktiken der Softwareentwicklung basiert. Alle Infrastruktur-Komponenten sollten als Code definiert und im Idealfall auch getestet werden.

Das bedeutet in der Praxis, dass man seine Infrastruktur entweder in einer Programmiersprache oder mittels Konfigurationen im JSON- oder YAML-Format beschreibt. Daraus entstehen dann Anweisungen, welche die Modifikationen an einer Umgebungsdefinition und der Version des Konfigurationsmodells durchführen. Ist diese Umgebung einmal eingerichtet, erfolgen Anpassungen nur noch an der Quelle und nicht mehr am Endpunkt. Die Konfiguration wird üblicherweise in einem Versionsverwaltungssystem wie Git abgelegt – so weiß man immer, wer, wann, was und warum er es geändert hat.

Das führt dazu, dass man Infrastruktur in einer konsistenten, wiederholbaren und automatisierten Weise bereitstellen kann. Damit vermeidet man Chaos und Fehler und spart einen Haufen Geld.


Die wesentlichen Vorteile gegenüber manuellen Konfigurationen sind:

1

**Kosteneinsparung und Beschleunigung**

Mit IaC können komplexe Infrastrukturen in wenigen Befehlen automatisch bereitgestellt und verwaltet werden. Dadurch reduzieren sich manuelle, zeitaufwendige Aufgaben. Will man Anpassungen an der Infrastruktur machen, ist dies in wenigen Minuten erledigt. Gleichzeitig ist es für neue Teammitglieder möglich, sich schnell einzuarbeiten und den Überblick zu behalten. So kann man kurzfristig und schnell auf neue Anforderungen reagieren und neue Produkte oder Services schnell am Markt platzieren.

2

### Konsistenz und Wiederholbarkeit

IaC bringt eine hohe Konsistenz mit sich, deren großes Potenzial sich vor allem nach einiger Zeit zeigt. Einmal erstellte und getestete Vorlagen können immer wieder verwendet werden, wodurch das Risiko von Abweichungen und menschlichen Fehlern drastisch reduziert wird. Die Bereitstellung und Anpassung der Infrastruktur wird vorhersehbar und zuverlässig. Infrastrukturen können in verschiedenen Umgebungen (z.B. Entwicklung, Test, Produktion) konsistent und wiederholt bereitgestellt werden. Dazu verringert sich die Abhängigkeit von einzelnen Teammitgliedern, da das Wissen nicht nur bei wenigen Personen zentralisiert ist.

3

### Skalierbarkeit

Mit IaC können Ressourcen leicht hinzugefügt oder entfernt werden, um sich an wechselnde Anforderungen anzupassen. Dies kann relevant werden, wenn man die Services oder digitalen Produkte schnell skalieren will bzw. noch nicht absehbar ist, wie hoch die Nachfrage sein wird.
Dies ist besonders wertvoll in Umgebungen mit variabler Last, wie bei Webanwendungen, die plötzlichen Traffic-Spitzen standhalten müssen. Zudem gewährleistet die konsistente und wiederholbare Natur von IaC, dass die Skalierung ohne Qualitätsverlust oder Abweichungen in der Konfiguration erfolgt.

4

### Versionierung

Mit IaC können Änderungen an der Infrastruktur genau nachverfolgt werden. Jede Version gibt einen klaren Überblick über die vorgenommenen Anpassungen, wodurch die Ursachenanalyse bei Problemen erleichtert wird. Außerdem ermöglicht die Versionierung "Rollbacks", also ein schnelles und sicheres Zurücksetzen auf eine vorherige, stabile Version. Dies reduziert Ausfallzeiten und Risiken. Teams können gleichzeitig an verschiedenen Teilen der Infrastruktur arbeiten, ohne sich gegenseitig zu beeinflussen. Änderungen können in separaten Versionen gespeichert und später zusammengeführt werden, was die Teamarbeit effizienter macht.

5

### Sicherheit und Compliance

Aus Sicherheits- und Compliance-Gesichtspunkten bietet IaC entscheidende Vorteile. Durch die hohe Standardisierung und Verwendung von IaC-Vorlagen wird sichergestellt, dass alle Infrastrukturelemente nach denselben Sicherheitsstandards und -richtlinien bereitgestellt werden. IaC-Tools können außerdem so konfiguriert werden, dass sie automatische Sicherheits- und Compliance-Überprüfungen durchführen, bevor Änderungen angewendet werden. Dies stellt sicher, dass keine unsicheren Konfigurationen in die Produktion gelangen.

Für Audits und Nachverfolgbarkeit sind alle Änderungen in IaC-Versionen dokumentiert. Und bei Sicherheitsvorfällen ermöglicht IaC eine schnelle Reaktion, indem bekannte sichere Konfigurationen sofort wiederhergestellt oder unsichere Komponenten isoliert werden können.

## Exkurs: Unterscheidungskriterien für IaC-Tools

Heutzutage steht eine breite Palette von IaC-Tools zur Verfügung. Diese können in verschiedenen Phasen und Szenarien des Betriebs von Anwendungen helfen. Unterscheidungskriterien für IaC-Tools können sein:

- Code-Logik: Imperativ vs. deklarativ
- Infrastruktur-Typ: Veränderbar vs. Unveränderbar
- Funktionsumfang: Bereitstellung (Provisionisierung) der Infrastruktur (z.B. VMs) vs. Konfiguration der Infrastruktur (Configuration Management des Betriebssystems)

## Code Logik: Imperativ vs. deklarativ

1

### Imperativer Ansatz

Beim imperativen Ansatz von IaC beschreibt man im Code den genauen Weg zur Erreichung des gewünschten Infrastrukturzustands. Man legt spezifische Befehle und Schritte fest, die ausgeführt werden müssen, um die Infrastruktur in den gewünschten Zustand zu versetzen.

**Vorteile**: Man bekommt eine hohe Kontrolle über den Bereitstellungsprozess, da jedes Detail manuell festgelegt wird.

**Nachteile**: Dieser Ansatz ist zeitaufwendiger und anfälliger für Fehler, da jedes Element und jeder Schritt manuell definiert werden muss. Außerdem kann es herausfordernd sein, den Überblick über den gesamten Zustand der Infrastruktur zu behalten – besonders in großen und komplexen Umgebungen. Beim imperativen Ansatz ist zu beachten, dass man sich selbst darum kümmern muss, dass Ressourcen nicht dupliziert angelegt werden.

2

### Deklarativer Ansatz

Im Gegensatz zum imperativen Ansatz, beschreibt man beim deklarativen Ansatz den gewünschten Endzustand der Infrastruktur. Die spezifischen Schritte und Befehle, die notwendig sind, um diesen Zustand zu erreichen, werden vom jeweiligen IaC-Tool automatisch abgeleitet und ausgeführt. Populäre deklarative IaC-Tools sind Terraform, AWS CloudFormation oder Puppet.

**Vorteile**: Dieser Ansatz ist oft einfacher und schneller umzusetzen, da Teams sich nicht um die Details des „Wie“ kümmern müssen, sondern sich auf das „Was“ konzentrieren können. Außerdem bekommt man eine konsistente Infrastruktur, da das IaC-Tool sicherstellt, dass die Umgebung immer dem definierten Zustand entspricht.

**Nachteile**: Mit dem deklarativen Ansatz kann es schwieriger sein, Probleme zu diagnostizieren und zu beheben, da die internen Abläufe des IaC-Tools abstrahiert sind und nicht direkt eingesehen werden können.

## Infrastruktur Typ: Veränderbar vs. Unveränderbar

1

### Veränderbare Infrastruktur

Veränderbare Infrastruktur bezieht sich auf eine Infrastruktur, bei der Änderungen und Updates direkt auf den bestehenden Servern und Systemen vorgenommen werden können. Tool-Beispiele in diesem Bereich sind Ansible, Puppet oder Chef.

**Vorteile**: Es ist möglich, schnelle Änderungen vorzunehmen.

**Nachteile**: Veränderbare Infrastruktur kann zu Inkonsistenzen führen, wenn nicht alle Systeme gleichzeitig oder auf die gleiche Weise aktualisiert werden.

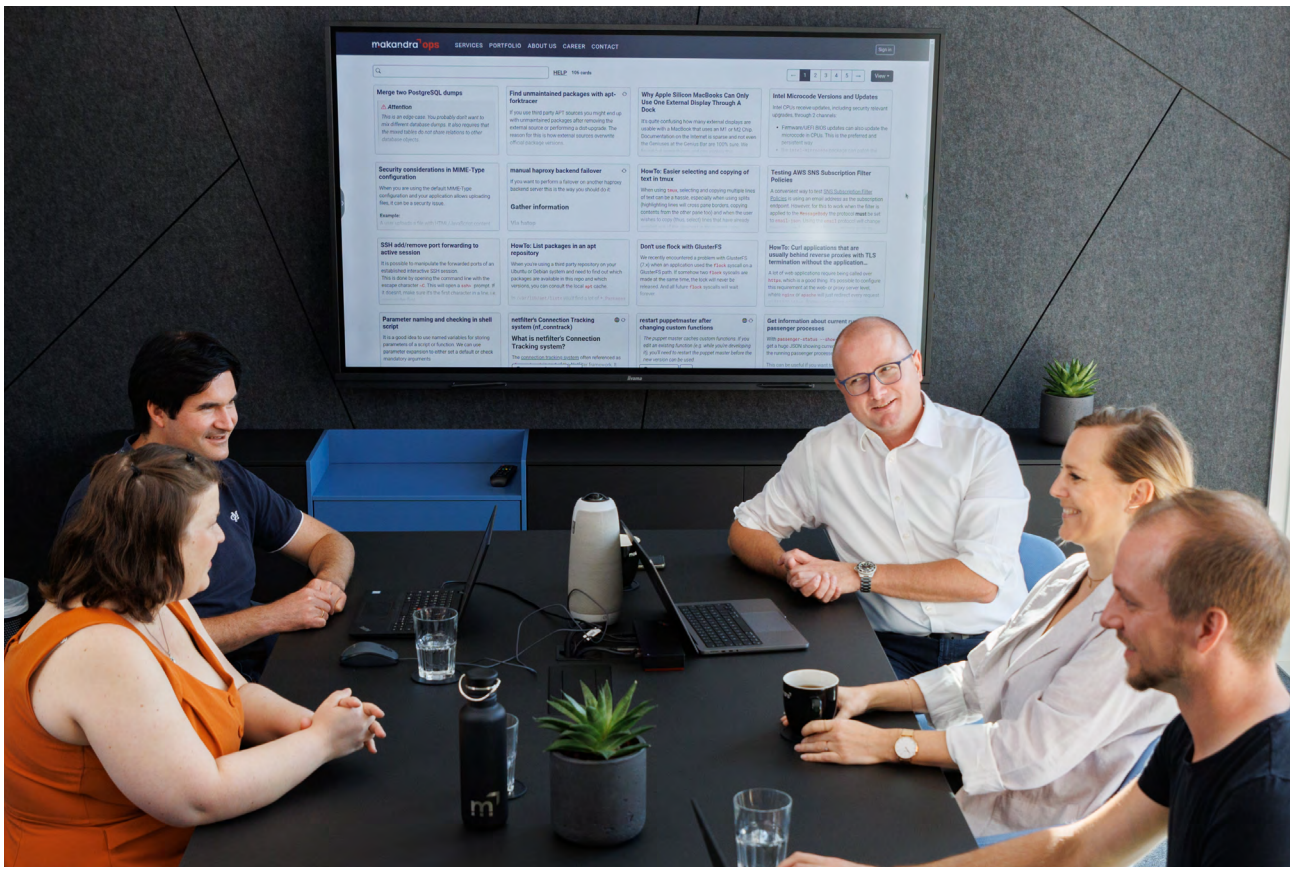
2

### Unveränderbare Infrastruktur

Unveränderbare Infrastruktur ist dagegen ein Ansatz, bei dem Änderungen nicht direkt auf bestehenden Systemen vorgenommen werden. Stattdessen wird bei jeder Änderung oder Aktualisierung eine neue Instanz der Ressource erstellt und die alte Instanz ersetzt. Dieser Ansatz wird üblicherweise mit virtuellen Maschinen und fertigen Disk Images oder Containern umgesetzt.

**Vorteile**: Die Wahrscheinlichkeit, dass es zu einem Konfigurationsdrift oder Inkonsistenzen kommt, ist sehr gering, da Änderungen immer auf einer “sauberen” Instanz vorgenommen werden.

**Nachteile**: Es kann sein, dass es mehr Ressourcen und Zeit in Anspruch nimmt, da für jede Änderung neue Instanzen erstellt werden müssen.



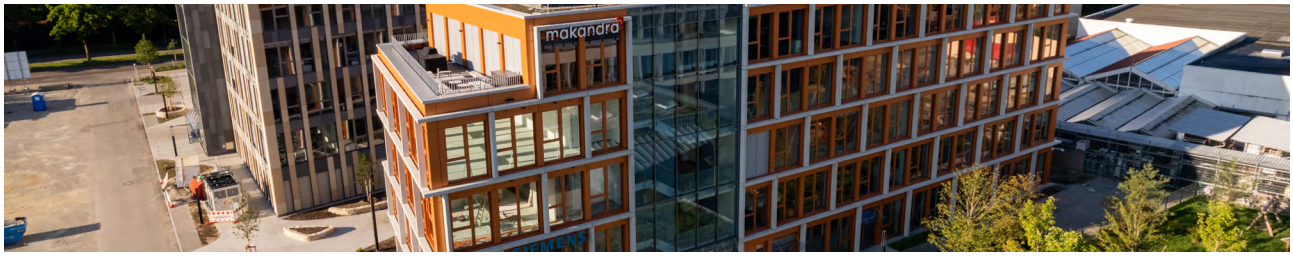
## Funktionsumfang: Bereitstellung vs. Konfiguration von Infrastruktur

Mittlerweile sind die meisten IaC-Tools sehr mächtig in Bezug auf ihren Funktionsumfang. Dennoch eignen sich einige eher für die Bereitstellung (Provisioning) der Infrastruktur und andere eher für die Einrichtung (Configuration) der Infrastruktur. Letzteres wird auch als Configuration Management bezeichnet.

### Bereitstellung der Infrastruktur

Die Bereitstellung der Infrastruktur bezieht sich auf den Prozess des Einrichtens und der Bereitstellung der physischen oder virtuellen Hardware-Ressourcen, die für den Betrieb von Anwendungen und Diensten erforderlich sind. Dies kann Server, Netzwerkkomponenten, Speicher und andere Hardware-Elemente umfassen. Beispielsweise kümmert man sich hier um die Einrichtung von Netzwerkverbindungen, Zuweisung von IP-Adressen, Konfiguration von Firewalls und Load Balancern sowie Sicherstellung der Netzwerksicherheit.

Die verbreitetsten Tools für das Provisioning von Infrastruktur sind Terraform, Pulumi, AWS CloudFormation und der Azure Resource Manager.




### Konfiguration der Infrastruktur

Die Konfiguration der Infrastruktur ist ein entscheidender Schritt im Infrastrukturmanagement, der auf das Provisioning folgt. Hierbei geht es darum, die bereitgestellten Hardware- und Software-Ressourcen so einzurichten, dass sie optimal zusammenarbeiten und die Anwendungen und Dienste unterstützen, die auf ihnen ausgeführt werden sollen. Dieser Prozess umfasst eine Vielzahl von Aufgaben und Überlegungen, unter anderem:

- die Einrichtung von Benutzerkonten
- die Konfiguration von Netzwerkeinstellungen
- die Installation von Softwarepaketen
- die Anwendung von Sicherheitspatches und Updates

Oft ist es notwendig, neben der bereits bestehenden Netzwerkkonfiguration während des Provisionings noch zusätzliche Netzwerkeinstellungen zu definieren, die auf Betriebssystem- oder Anwendungsebene vorgenommen werden müssen. Zum Beispiel:

- die Konfiguration von Firewall-Regeln
- die Konfiguration von Load Balancern
- die Einrichtung von VPN-Verbindungen

Oft eingesetzte Werkzeuge für das Configuration Management sind Ansible, Puppet und Chef.

# Übersicht der populärsten IaC-Tools

Nachdem wir nun die Vorteile von IaC kennen, zeigen wir nachfolgend die populärsten Tools.

Wir beginnen mit Tools, die sich besonders für die **Bereitstellung der Infrastruktur** eignen: Terraform, AWS CloudFormation und Pulumi.

1

### Terraform

Terraform, entwickelt von HashiCorp, ist ein Open-Source-Tool, das es ermöglicht, Ressourcen in verschiedenen Cloud-Providern wie AWS, Azure und Google Cloud zu definieren und bereitzustellen. Mit einer deklarativen Konfigurationssprache (HashiCorp Configuration Language (HCL)) beschreiben Nutzer die gewünschte Infrastruktur. Terraform kümmert sich um die Erstellung und Verwaltung dieser Ressourcen. Es wird häufig eingesetzt, um Netzwerke, virtuelle Maschinen, Managed Services und andere Infrastrukturkomponenten zu provisionieren und zu verwalten. Terraform wird von einer großen Community unterstützt.

2

### AWS CloudFormation

AWS CloudFormation ist ein Service von Amazon-Web-Services und ermöglicht ebenfalls die Infrastructure-as-Code-Bereitstellung. Im Gegensatz zu Terraform ist es speziell auf AWS-Ressourcen zugeschnitten. Nutzer definieren ihre AWS-Infrastruktur in JSON- oder YAML-Dateien, und CloudFormation erstellt und verwaltet die Ressourcen entsprechend. Ein typisches Einsatzgebiet ist die Automatisierung der Bereitstellung von EC2-Instanzen, S3-Buckets und Lambda-Funktionen. Der größte Vorteil von CloudFormation ist seine tiefe Integration in die AWS-Umgebung, was eine sehr detaillierte Konfiguration von AWS-Ressourcen ermöglicht. Die Syntax der Vorlagen kann jedoch – speziell bei einer gewachsenen Infrastruktur – komplex und schwer zu verstehen sein.

3

### Pulumi

Pulumi ist ein noch sehr junges Infrastructure-as-Code-Tool, das die gleichen Funktionen wie Terraform bietet, jedoch den Anwendern erlaubt, ihre Bereitstellungslogik in gängigen Programmiersprachen wie Go, TypeScript, Python oder .NET zu formulieren, anstelle von spezifischen Konfigurationssprachen wie YAML oder HCL. Pulumi unterstützt eine Vielzahl von Cloud-Anbietern, einschließlich AWS, Azure und Google Cloud sowie Kubernetes, was eine flexible und anbieterübergreifende Infrastrukturautomatisierung ermöglicht. Allerdings kann die Verwendung von allgemeinen Programmiersprachen auch zu Komplexität führen, insbesondere wenn Entwickler nicht mit Best Practices für IaC vertraut sind. Zudem ist Pulumi ein relativ junges Tool im Vergleich zu etablierten Lösungen wie Terraform, was bedeutet, dass die Community und das Ökosystem noch wachsen.

Nachdem die Infrastruktur aufgesetzt ist, gilt es, soweit erforderlich, sie zu konfigurieren, um die gewünschten Funktionen auszuführen. Dies trifft primär auf virtuelle Maschinen zu. Verwendet man Managed Services in der Cloud oder Container, wurde die Konfiguration schon im vorherigen Schritt festgelegt. Traditionell ist die Konfiguration von Linux- oder Windows-Systemen oft von manuellen Eingriffen und vagen Dokumentationen geprägt. Nachfolgende Werkzeuge haben sich bewährt, um die **Konfiguration und das Management von Servern** zu automatisieren und zu vereinfachen.

4

### Chef

Chef – entwickelt in Ruby – ermöglicht es Nutzern, ihre Infrastruktur in Form von „Rezepten" und „Kochbüchern" zu definieren, die die gewünschten Zustände der Systeme beschreiben. Diese Skripte werden auf den Zielservern ausgeführt, um die Konfigurationen anzupassen und sicherzustellen, dass sie den definierten Anforderungen entsprechen. Ein typisches Einsatzgebiet für Chef könnte die Automatisierung der Konfiguration von Webservern, Datenbanken oder der Installation von Software-Paketen sein. Chef bietet eine hohe Flexibilität und Erweiterbarkeit, kann jedoch aufgrund seiner DSL (Domain Specific Language) und der Vielzahl an Konzepten eine steile Lernkurve aufweisen.

5

### Puppet

Puppet, ebenfalls in Ruby entwickelt, bietet eine deklarative Sprache, um den gewünschten Zustand der Infrastruktur zu beschreiben. Es nutzt einen Client-Server-Ansatz, bei dem der Puppet-Server die Konfigurationen verwaltet und die Agenten auf den Zielservern diese Konfigurationen anwenden. Puppet wird oft in großen, komplexen Umgebungen eingesetzt, wo es eine konsistente und wiederholbare Serverkonfiguration sicherstellt. Ein Vorteil von Puppet ist seine starke Community und die breite Palette an verfügbaren Modulen, die die Automatisierung von gängigen Aufgaben erleichtern. Zur Unterscheidung: Chef bietet eine flexiblere, script-basierte Herangehensweise. Puppet besticht durch seine deklarative Sprache und seinen strukturierten Ansatz. Als Alternative zum Client-Server-Betrieb kann auch Puppet-Bolt verwendet werden. Damit erhält man ein ähnliches Verhalten wie bei Ansible.

6

### Ansible

Ansible, das jünger und neuer ist als Chef und Puppet, basiert ebenfalls auf Open-Source-Prinzipien. Es ist derzeit sehr populär, weil es sehr einfach und zugänglich ist und damit eine leistungsstarke und benutzerfreundliche Option für Teams bietet, die eine schnelle und effiziente Lösung für ihre Automatisierungsbedürfnisse suchen. Entwickelt in Python, ermöglicht Ansible Nutzern, ihre Infrastruktur durch einfache YAML-Dateien, sogenannte Playbooks, zu definieren und zu verwalten. Diese Playbooks beschreiben die gewünschten Zustände der Systeme und die Schritte, die notwendig sind, um diese Zustände zu erreichen. Ein großer Vorteil von Ansible ist sein agentenloser Ansatz, bei dem keine Software auf den Zielservern installiert werden muss, da die Kommunikation und Konfiguration über SSH erfolgt. Dies vereinfacht die Einrichtung und Wartung erheblich und reduziert den Overhead auf den verwalteten Systemen.

Ansible bietet eine umfangreiche Sammlung von Modulen und eine aktive Community, die ständig neue Playbooks und Lösungen für gängige Automatisierungsaufgaben bereitstellt. Als Nachteil kann die Notwendigkeit des vollständigen Beschreibens der Infrastruktur in YAML gesehen werden. Komplexe Abläufe und Logik lassen sich dabei nicht so gut darstellen und sind gegebenenfalls nicht mehr so gut lesbar.

# Deep Dive Terraform

Im begrenzten Rahmen dieses Whitepaper können wir nicht alle IaC-Werkzeuge vollumfänglich vorstellen. Deshalb entscheiden wir uns für das derzeit beliebteste und verbreitetste Provisioning-Tool, das auch als "Standard-Tool für IaC" bezeichnet wird: Terraform. Vor allem für Einsteiger bietet es eine gute Möglichkeit, IAC professionell umzusetzen.

In Terraform wird die Infrastruktur durch Konfigurationsdateien dargestellt, welche Terraform anschließend nutzt, um die erforderlichen Ressourcen bei Cloud-Anbietern oder auf anderen Plattformen zu installieren. Ein wesentliches Prinzip bei Terraform ist, dass eine bestimmte Konfiguration wiederholt angewendet werden kann und das Resultat konstant bleibt, ohne ungewollte Nebeneffekte zu verursachen. Die HashiCorp Configuration Language (HCL), in der man die Konfigurationsdateien in Terraform schreibt, wurde speziell entwickelt, um Infrastruktur Ressourcen auf eine klare und knappe Art und Weise zu beschreiben. Terraform funktioniert auf Basis einer reinen Client-Struktur, wodurch keine weiteren Konfigurationsmanagement-Einstellungen auf einem Server notwendig sind.

## Kernkonzepte von Terraform

Terraform basiert auf einer Reihe von Kernprinzipien, die essenziell sind, um das Tool richtig zu verstehen und effizient einsetzen zu können.

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Provider | Ressourcen | State | Module | Variablen & Outputs |

### Provider

1

Provider sind Plugins, die Terraform mit verschiedenen Cloud-Anbietern, SaaS-Anbietern und anderen APIs verbinden. Sie sind verantwortlich für die Erstellung, Aktualisierung und Löschung von Ressourcen. Beispiele für Provider sind AWS, Azure, Google Cloud Platform und viele mehr. Als Nutzer muss man die Provider konfigurieren, um Zugang zu diesen Diensten zu erhalten und die Ressourcen innerhalb dieser Dienste zu managen. Grundsätzlich kann jeder selbst einen Provider in Go entwickeln und in Terraform einbinden. Die einzige Voraussetzung ist, dass das Zielsystem eine CRUD API bietet.

### Ressourcen

2

Ressourcen sind die Kernbausteine in Terraform und repräsentieren die verschiedenen Komponenten der Infrastruktur, wie z.B. virtuelle Maschinen, Netzwerke oder Datenbanken. Jede Ressource wird durch einen Provider verwaltet und durch Konfigurationsdateien erstellt, modifiziert oder gelöscht.

3

### State

Der Zustand (“State”) ist eine kritische Komponente in Terraform, die den aktuellen Status der verwalteten Infrastruktur festhält. Der State stellt eine Verbindung der API-Objekte des Zielsystems mit den definierten Ressourcen her. Erstellt man z.B. eine EC2-Instanz bei AWS, erhält diese eine zufällig generierte ID. Terraform vermerkt im State welcher Ressource im Code diese ID zugeordnet ist. Außerdem nutzt es den State, um Änderungen und Aktualisierungen effizient durchzuführen und um sicherzustellen, dass die reale Infrastruktur mit dem im Code definierten Zustand übereinstimmt. Zudem können Ressourcen automatisch gelöscht werden, wenn diese aus dem Code entfernt werden.

4

### Module

Module ermöglichen die Wiederverwendung und Organisation von Terraform-Code. Es handelt sich um einen Ordner mit Terraform-Vorlagen, in dem alle Terraform-Konfigurationen und Ressourcen enthalten sind. Diese können auch für andere Terraform-Projekte verwendet werden.

5

### Variablen und Outputs

Variablen ermöglichen die Parametrisierung von Terraform-Konfigurationen, sodass Benutzer Eingaben bereitstellen können, ohne den Code direkt zu ändern. Outputs sind eine Möglichkeit, Daten aus Terraform-Konfigurationen zu exportieren, um sie in anderen Teilen der Infrastruktur oder in anderen Tools zu verwenden.

## Zentrale Befehle von Terraform

Die wichtigsten Befehle bei Terraform sind `init`, `plan`, `apply` und `destroy`.

| | |
|---|---|
| `terraform init` | Initialisiert das Arbeitsverzeichnis, in dem sich alle Konfigurationsdateien befinden. |
| `terraform plan` | Mit diesem Befehl kann man überprüfen, welche Änderungen Terraform an der Infrastruktur vornehmen wird, bevor man diese tatsächlich anwendet. |
| `terraform apply` | Ist man mit den geplanten Änderungen zufrieden, führt man `terraform apply` aus, um die Infrastruktur zu erstellen oder anzupassen. |
| `terraform destroy` | Wenn man die erstellte Infrastruktur nicht mehr benötigt, kann man sie mit `terraform destroy` wieder entfernen. |

# Erste Schritte mit IaC

Nachdem wir nun einige IaC-Tools kennengelernt haben, steht einem ersten Ausprobieren nichts mehr im Wege. Wie aber sollte man am besten vorgehen?

Hier sind die wichtigsten Schritte, wie Sie Ihre Reise zu IaC starten können.

## 1 Ziele und Anforderungen an IaC definieren

Bevor man sich tiefer mit IaC auseinandersetzt, sollte man sich die Zeit nehmen, um Ziele und Anforderungen präzise zu definieren. Überlegen Sie genau, was Sie mit der Einführung von IaC erreichen möchten. Das kann beispielsweise die Automatisierung wiederkehrender Aufgaben, die Verbesserung der Konsistenz über verschiedene Umgebungen hinweg oder die Beschleunigung der Provisioning-Prozesse umfassen. Durch die Festlegung klarer Ziele und Anforderungen stellen Sie sicher, dass Ihr IaC-Ansatz auf die spezifischen Herausforderungen und Bedürfnisse Ihres Unternehmens zugeschnitten ist. Es hilft, wenn alle Teammitglieder die gleichen Ziele vor Augen haben und mit einem gemeinsamen Verständnis starten.

## 2 Bestandsaufnahme Ihrer Infrastruktur Komponenten

Im zweiten Schritt sollten Sie sich mit den derzeitigen Komponenten Ihrer Infrastruktur beschäftigen. Hierbei geht es darum, jedes einzelne Element, das in Ihrer Umgebung vorhanden ist, genau zu identifizieren und zu dokumentieren. Dazu zählen Server, Netzwerke, Speichersysteme, Anwendungen und alle weiteren Ressourcen, die für den Betrieb Ihrer IT-Landschaft erforderlich sind. Es ist wichtig, dass Sie die derzeitigen Funktionen, Abhängigkeiten und Konfigurationen verstehen, bevor Sie in die Automatisierung starten. Dadurch sind Sie in der Lage, Ihre IaC-Skripte so zu gestalten, dass sie exakt auf Ihre spezifischen Anforderungen und Rahmenbedingungen zugeschnitten sind.

## 3 Wahl des passenden IaC-Tools

Nachdem Sie Ihre Ziele und Anforderungen definiert und Ihre Infrastrukturkomponenten identifiziert haben, steht als nächster Schritt die Auswahl des geeigneten IaC-Tools an. Berücksichtigen Sie dabei vor allem die spezifischen Bedürfnisse und Anforderungen Ihres Unternehmens. Achten Sie auch auf die Lernkurve des Tools und die Verfügbarkeit von Ressourcen und Community-Unterstützung. Ein Tool mit einer aktiven Community und umfangreichen Lernmaterialien kann den Einstieg und die fortlaufende Nutzung erheblich erleichtern. Außerdem lohnt es sich, zu evaluieren, ob das Tool auch zukunftsfähig ist, also ein Mindestmaß an Skalierbarkeit und Flexibilität mitbringt.

## 4 Lernen Sie die Tool-spezifischen Sprachen und Konzepte

Nachdem Sie sich für ein IaC-Tool entschieden haben, ist der nächste entscheidende Schritt, sich mit der spezifischen Sprache und den Konzepten dieses Tools vertraut zu machen. Jedes Tool verwendet seine eigene Konfigurationssprache und hat eigene Arbeitsweisen, die es zu verstehen gilt.

Nehmen Sie sich die Zeit, die Dokumentation des Tools gründlich durchzuarbeiten und sich mit der Syntax und den Funktionen der Sprache auseinanderzusetzen. Nutzen Sie auch Online-Ressourcen, Tutorials und Community-Foren, um zusätzliche Hilfe und Einblicke zu erhalten.

## 5 Praxiserfahrung sammeln - Automatisieren einer kleinen Aufgabe

Jetzt ist es an der Zeit, praktische Erfahrungen zu sammeln. Starten Sie mit einer überschaubaren und gut definierten Aufgabe, um den Einstieg zu erleichtern. Dies könnte beispielsweise die Automatisierung der Provisionierung eines Webservers, die Konfiguration von Netzwerkeinstellungen oder die Installation einer bestimmten Software sein. Wählen Sie eine Aufgabe, die einen klaren Anfang und ein klares Ende hat und die Sie gut verstehen.

Nutzen Sie die Gelegenheit, mit verschiedenen Konfigurationsoptionen zu experimentieren und verschiedene Ansätze auszuprobieren. Dokumentieren Sie Ihre Erfahrungen und die Lösungen für aufgetretene Probleme, um Ihr Wissen zu festigen und eine Ressource für zukünftige Aufgaben zu schaffen.

## 6 Teste Sie Ihren Code

Stellen Sie sicher, dass jede Funktion und jedes Modul wie erwartet arbeitet. Dies hilft, Fehler frühzeitig zu erkennen und die Stabilität Ihres Codes zu gewährleisten. Achten Sie darauf, dass Ressourcen korrekt miteinander verknüpft sind und dass die Konfigurationen in der realen Umgebung wie erwartet funktionieren.

## 7 Planen sie eine Hierarchie/ Architektur für ihren Code

Insbesondere wenn die Komplexität in der Infrastruktur steigt oder schon hoch ist, ist es entscheidend, den Code modular abzulegen und sich zu überlegen, wie man die Duplizierung von Code vermeiden kann (DRY-Konzept – Don’t Repeat Yourself). Außerdem sollte die Hierarchie und Strukturierung des Codes und der Module einen einfachen und übersichtlichen Einstieg in die Codebasis liefern. Einzelne Komponenten und deren Konfiguration müssen einfach zu finden sein. Siehe hierzu auch die Best Practices.


## 8 Verwenden Sie ein Version-Control-System

Für Ihre ersten IaC-Versuche sollten Sie auf jeden Fall ein Version-Control-System (VCS) wie Git, Mercurial oder SVN nutzen. Es ermöglicht Ihnen, Änderungen an Ihrem Code nachzuverfolgen, verschiedene Versionen zu speichern und bei Bedarf zu früheren Versionen zurückzukehren. Außerdem erleichtert es die Zusammenarbeit im Team, da mehrere Personen gleichzeitig an verschiedenen Teilen des Codes arbeiten können, ohne sich gegenseitig zu stören. Für eine etwaige Fehlerbehebung ist dies auch von Vorteil, da Sie leicht sehen können, welche Änderungen wann und von wem vorgenommen wurden. Außerdem legt es den Grundstein für die Implementierung von CI/CD-Pipelines, da es eine automatisierte Möglichkeit bietet, Ihren Code von der Entwicklung über Tests bis hin zur Produktion zu bewegen.

## 9 Automatisieren Sie das Provisioning (CI/CD-Pipeline)

Im letzten Schritt sollten Sie Ihren Code mithilfe einer CI/CD-Pipeline auf Ihrer Infrastruktur deployen. Dies stellt sicher, dass Ihre Infrastruktur stets aktuell ist und mit Ihrem Code synchronisiert bleibt. Geläufige Tools für CI/CD-Pipelines sind Jenkins, GitLab CI, GitHub Actions, Travis CI oder CircleCI. Diese automatisieren Ihren Deployment Prozess und stellen sicher, dass Ihre Infrastruktur konsistent in verschiedenen Umgebungen eingesetzt wird.

# Best Practices für IaC

Sie kennen nun die wichtigsten IaC-Tools und wissen, wie man die ersten Schritte mit IaC macht. Nachfolgend möchten wir Sie noch auf Best Practices hinweisen, die wir in unserem Praxisalltag gelernt haben.

1

### Alles Coden – Konfigurationsdateien als Wahrheitsquelle

Haben Sie sich für einen IaC-Ansatz entschieden, dann sollten Sie auch versuchen, diesen möglichst konsequent durchzuziehen.
Schreiben Sie alle Ihre Infrastruktur-Spezifikationen explizit in Konfigurationsdateien – egal ob es sich um AWS CloudFormation-Vorlagen, Chef-Rezepte, Ansible-Playbooks oder jedes beliebige andere IaC-Tool handelt. Diese Konfigurationsdateien repräsentieren die einzige Wahrheitsquelle Ihrer Infrastruktur-Spezifikationen und beschreiben genau, welche Komponenten Sie verwenden werden, wie sie miteinander in Beziehung stehen und wie die gesamte Umgebung konfiguriert ist.
Die Infrastruktur kann dann schnell und nahtlos bereitgestellt werden, und niemand muss sich mehr manuell in einen Server einloggen, um Anpassungen vorzunehmen.

2

### Nutzen eines Versionskontrollsystems

Versionskontrolle bei IaC ist besonders wichtig, wenn es um Konfigurationsdateien geht. Dadurch können Sie problemlos Änderungen an Ihren Systemen verfolgen, verwalten und bei Bedarf wiederherstellen, was eine verbesserte Nachverfolgbarkeit und Sichtbarkeit bietet.
Besonders hilfreich ist dies, wenn man Probleme debuggen und diagnostizieren muss oder möglicherweise zu einer früheren Version zurückkehren will. Außerdem kann man automatisch Aktionen auslösen, wenn eine Änderung übernommen wird, was ein Schlüssel zu CI/CD-Pipelines ist.
Es gibt zahlreiche Tools für Versionskontrolle, Änderungsverfolgung und Quellcodeverwaltung auf dem Markt. Dazu gehören GitHub, GitLab, BitBucket, Azure DevOps Server usw.. Diese können mit dem jeweils bevorzugten IaC-Tool verbunden werden.

3

### Modulare Struktur

Theoretisch ist es natürlich möglich, eine einzige IaC-Datei zu schreiben, die alle Aspekte einer bestimmten Ressource konfiguriert, z.B. Betriebssystem, Benutzerkonten, Installation von Anwendungen, Netzwerkregeln etc.. Praktisch ist es aber sinnvoller, seinen IaC-Code in wiederverwendbare Module zu strukturieren. Durch diesen modularen Ansatz können Teams einen Teil einer Konfiguration leichter aktualisieren, ohne andere Teile zu berühren. Außerdem ist es dadurch möglich, einige IaC-Regeln über Ressourcen hinweg zu teilen, aber nicht andere. Ein Team könnte beispielsweise zwei Servergruppen haben, die dasselbe Betriebssystem installiert haben müssen, aber unterschiedliche Benutzerkonten benötigen. Modulare IaC-Regeln erleichtern die Bewältigung solcher Anwendungsfälle. Außerdem hat man einen besseren Überblick über seine Code-Struktur und es erlaubt eine schnellere und leichtere Fehlererkennung nach Änderungen.

4

### Unveränderliche Infrastruktur

Streben Sie eine unveränderliche Infrastruktur an, bei der Änderungen durch das Ersetzen von Ressourcen und nicht durch deren Änderung vorgenommen werden. Wenn Sie beispielsweise das Betriebssystem eines virtuellen Servers aktualisieren müssen, würden Sie den vorhandenen Server löschen und ihn durch ein neues Server-Image basierend auf einer aktualisierten Konfiguration ersetzen. Es kann natürlich sinnvoll sein, sehr kleine Änderungen wie zb. das Installieren einer neuen Bibliothek auf einem bereits vorhandenen Server oder das Vornehmen einer kleinen Änderung an der Netzwerkkonfiguration eines Routers direkt vorzunehmen.

Generell ist IaC jedoch zuverlässiger, wenn Teams eine Strategie für unveränderliche Infrastruktur verfolgen. Durch die Unveränderlichkeit der Infrastruktur wird Konsistenz gewährleistet, Konfigurations-Abweichungen werden vermieden und die Auswirkungen von undokumentierten Änderungen am Setup eingeschränkt.

5

### Continuous Testing/CI/CD

Eine der wichtigsten Best Practices bei IaC, die Infrastrukturteams von der Softwareentwicklung übernehmen können, ist effektives Testen. Tests sollten rigoros auf Ihre Infrastruktur-Konfigurationen angewendet werden, um sicherzustellen, dass es keine Probleme nach der Bereitstellung gibt. Je nach Bedarf können verschiedene Tests wie Integrations- oder Regressionstests durchgeführt werden. Diese Tests können automatisiert jedes Mal durchgeführt werden, wenn eine Änderung am Code vorgenommen wird.

Mit einem soliden CI-Prozess können diese Konfigurationsvorlagen mehrfach in verschiedenen Umgebungen (zb. Staging/Production) bereitgestellt werden. Entwickler können dann in jeder dieser Umgebungen mit konsistenten Infrastruktur-Konfigurationen arbeiten. Zusätzlich hilft ein gutes Monitoring. Dieses stellt auch außerhalb von Konfigurationsanwendungen sicher, dass sich die Infrastruktur so verhält, wie es erwartet wird.

6

### Keine Secrets in IaC

Manchmal müssen IaC-Tools auf sensible Informationen wie Passwörter oder Verschlüsselungsschlüssel zugreifen, um Ressourcen zu konfigurieren. Der einfachste Weg, Geheimnisse zu definieren, besteht darin, sie direkt in den IaC-Definitionen einzuschließen. Da IaC-Regeln jedoch Klartextdateien sind, ist diese Praxis aus Sicherheitssicht sehr riskant. Jeder, der Zugang zu den IaC-Dateien erhält, kann alle darin gespeicherten Passwörter oder andere sensible Daten lesen.

Dies ist daher unbedingt zu vermeiden. Eine bessere Praxis besteht darin, sensible Daten in einem Geheimnis-Manager zu speichern, wo sie sicher aufbewahrt und bei Bedarf von IaC-Tools abgerufen werden können. Einige IaC-Tools bieten ihre eigenen Tools zur Geheimnisverwaltung an, wie zum Beispiel Chef-vault im Fall von Chef. Sie können in der Regel auch Drittanbieter-Geheimnis-Manager wie CyberArk, Conjur oder ein cloudbasiertes Geheimnis-Tool wie AWS Secrets Manager in Verbindung mit IaC-Tools verwenden.

# Herausforderungen von IaC

Bei IaC gibt es allerdings auch gewisse Herausforderungen, auf die man sich vorbereiten sollte. Nachfolgend zeigen wir Ihnen die aus unserer Sicht wesentlichsten Herausforderungen, die wir regelmäßig in unserem Projektalltag mitbekommen – inklusive unserer jeweils vorgeschlagenen Lösungen dazu.

## Herausforderung 1: Hohe Lernkurve und fehlende Akzeptanz des Teams

Eine Einführung von IaC stellt für jedes Team erst mal eine Herausforderung dar. Eine Umstellung von manuellen Prozessen und traditionellen Infrastrukturmanagement-Praktiken auf eine automatisierte, codebasierte Umgebung erfordert nicht nur neue Fähigkeiten und Kenntnisse, sondern auch eine Veränderung der Denkweise und Arbeitskultur. Jeder im Team muss sich zwangsläufig mit Prinzipien der Softwareentwicklung, einschließlich Versionskontrolle, Code-Reviews und Continuous Integration/Continuous Deployment (CI/CD), vertraut machen. Dies kann eine hohe Lernkurve mit sich bringen und auf Widerstand im Team stoßen.

### Lösungsvorschläge

- **Weiterbildung**: Es ist sinnvoll, Ihr Team auf Schulungen und Weiterbildungen zu schicken, um es mit den notwendigen Fähigkeiten und Kenntnissen auszustatten.
- **Mentoring und Pair Programming**: Etablieren Sie ein Mentoring-System, bei dem erfahrene Teammitglieder ihr Wissen und ihre Erfahrungen mit IaC an weniger erfahrene Kollegen weitergeben. Pair Programming bietet sich ebenfalls an.
- **Klein anfangen**: Beginnen Sie mit einfachen Infrastrukturkomponenten und steigern Sie allmählich die Komplexität. Dieser Ansatz ermöglicht es dem Team, Erfahrung und Vertrauen zu sammeln, bevor es sich an komplexere Konfigurationen wagt.

## Herausforderung 2: Konfigurationsabweichungen (Configuration Drifts) durch manuelle Eingriffe

Besonders zu Beginn der IaC-Reise kann es vorkommen, dass Konfigurationsfehler durch manuelle Anpassungen von einzelnen Personen passieren. Beispielsweise kann eine Person noch nicht genau wissen, welche Codeänderungen für die Bereitstellung der Infrastruktur erforderlich sind, und könnte sich daher entscheiden, Änderungen manuell über die Konsole vorzunehmen. Dies führt dann natürlich dazu, dass das, was im Code definiert wurde, tatsächlich nicht dem entspricht, was bereitgestellt wurde. Und wenn Sie das nächste Mal den Code ausführen, könnte er versuchen, die manuell durchgeführte Änderung rückgängig zu machen und einen Ausfall verursachen.

### Lösungsvorschläge

- **Bewusstsein schaffen:** Es ist wichtig, das Team über die Konsequenzen manueller Eingriffe zu informieren und sie zu bitten, manuelle Änderungen über die Konsole zu vermeiden.
- **Strikte Prozesse für Änderungen:** Etablieren Sie klare Richtlinien, die manuelle Änderungen an der Infrastruktur außerhalb des IaC-Prozesses nicht gestatten. Stellen Sie sicher, dass alle Änderungen durch den IaC-Workflow gehen und im Code festgehalten werden.
- **Automatisierte Überwachung:** Implementieren Sie Tools, die die tatsächliche Infrastruktur mit dem IaC-Code abgleichen und Abweichungen automatisch erkennen. Bei Diskrepanzen sollten Alarme ausgelöst werden, die eine schnelle Reaktion ermöglichen.
- **Rollback-Strategien:** Entwickeln Sie klare Prozesse für den Fall, dass ein Configuration Drift auftritt. Dies sollte das Zurücksetzen auf einen bekannten guten Zustand beinhalten, der im IaC-Code definiert ist.

## Herausforderung 3: Konstante Aktualisierungen und Pflege

Eine der Herausforderungen, die Infrastructure as Code (IaC) mit sich bringt, ist der ständige Bedarf an Aktualisierungen. Dies ist vor allem bedingt durch die rasche Weiterentwicklung der Technologielandschaft. Beispielsweise können Cloud-Anbieter ihre APIs und Richtlinien ändern. Wenn dies geschieht, müssen auch die von Ihnen verwendeten IaC-Skripte und -Templates überarbeitet und aktualisiert werden, um sicherzustellen, dass Ihre bestehende Infrastruktur genau wie zuvor funktioniert. Das kann besonders knifflig sein, wenn Sie Open-Source-Tools verwenden, bei denen es zu Verzögerungen bei der Veröffentlichung von Änderungen kommen kann. Ohne regelmäßige Updates könnten veraltete Skripte zu Inkompatibilitäten oder Sicherheitslücken führen, die die Stabilität und Sicherheit der IT-Infrastruktur gefährden. Daher ist es wichtig, dass Unternehmen Prozesse und Ressourcen bereitstellen, um ihre IaC-Konfigurationen kontinuierlich zu pflegen und zu verbessern.

### Lösungsvorschläge

- **Aufwände frühzeitig planen:** Für die Pflege und konstante Wartung der IaC-Konfigurationen sollte man frühzeitig Prozesse und Ressourcen bereitstellen
- **Automatisierte Update-Prozesse:** Implementieren Sie automatisierte Prozesse, die regelmäßig nach Updates für die verwendeten IaC-Tools und Abhängigkeiten suchen und diese, wenn möglich, automatisch anwenden.
- **Abonnieren von Benachrichtigungsdiensten:** Nutzen Sie Benachrichtigungsdienste der Cloud-Anbieter oder Open-Source-Projekte, um über Änderungen an APIs oder Policies informiert zu werden, damit Sie proaktiv handeln können.

## Herausforderung 4: Automatisiertes Ausrollen von Änderungen benötigt ein ausgereiftes Test- und Abnahme-Konzept

Eine große Herausforderung bei der Anwendung von IaC ist es, umfassende Test- und Validierungsprozesse einzurichten und konsequent einzuhalten. Diese müssen sich auf alle Infrastruktur-Ebenen erstrecken – einschließlich Test-, Staging- und Produktionsumgebungen. Darüber hinaus muss ein Abnahmeprozess etabliert werden, der sicherstellt, dass alle Änderungen den Anforderungen der zuständigen Teams entsprechen und keine unerwünschten Nebeneffekte mit sich bringen. Dies ist insbesondere deshalb wichtig, da durch die Automatisierung des Deployments von Änderungen das Risiko besteht, dass Fehler schnell auf das gesamte System übertragen werden. Ein einzelner Fehler, der in einer manuellen Umgebung möglicherweise nur ein System beeinträchtigt hätte, hat nun das Potenzial, die gesamte IT-Infrastruktur zu beeinträchtigen.

### Lösungsvorschlag

- **Mehrstufige Deployment-Pipelines:** Richten Sie mehrstufige Deployment-Pipelines ein, die Änderungen durch verschiedene Umgebungen (Entwicklung, Test, Staging, Produktion) schleusen, wobei in jeder Stufe spezifische Tests und Qualitätskontrollen durchgeführt werden.

## Herausforderung 5: Abhängigkeiten und Wechselwirkungen

Infrastruktur-Ressourcen weisen oft komplexe Abhängigkeiten und Wechselwirkungen auf. In vielschichtigen Systemen, in denen verschiedene Komponenten und Dienste miteinander interagieren, müssen Entwickler und Betriebsteams sicherstellen, dass Änderungen an einer Stelle nicht unbeabsichtigte Konsequenzen an anderer Stelle nach sich ziehen. Dies erfordert ein tiefes Verständnis der Infrastruktur und eine sorgfältige Planung, um sicherzustellen, dass alle Teile des Systems harmonisch zusammenarbeiten. Es ist notwendig, die Abhängigkeiten zwischen Ressourcen genau zu definieren und zu dokumentieren, um Konflikte und Ausfälle zu vermeiden. Automatisierte Tests und eine robuste Versionskontrolle können dabei helfen, die Integrität der Infrastruktur bei Änderungen zu wahren.

### Lösungsvorschläge

- **Infrastruktur-Orchestrierung:** Nutzen Sie Infrastruktur-Orchestrierungstools wie Terraform oder AWS CloudFormation, die Funktionen zur Abhängigkeitsverwaltung bieten. Diese Tools ermöglichen es Ihnen, die Beziehungen zwischen Ressourcen explizit zu definieren und Abhängigkeiten automatisch zu verwalten.
- **Automatisierte Abhängigkeitsverfolgung:** Setzen Sie auf Infrastructure-as-Code-Frameworks, die automatisierte Funktionen zur Abhängigkeitsverfolgung bieten. Diese Frameworks analysieren den Code und erstellen Abhängigkeitsgraphen, um beim Verständnis und der Verwaltung von Abhängigkeiten zu helfen.

## Herausforderung 6: Das Gleichgewicht zwischen Sicherheit und Benutzerfreundlichkeit

Die korrekte Handhabung von Geheimnissen, wie Passwörtern, API-Schlüsseln und Zertifikaten, ist eine wesentliche Herausforderung. In IaC-Umgebungen, wo Infrastrukturkonfigurationen als Code definiert und oft in Versionskontrollsystemen gespeichert werden, besteht das Risiko, dass sensible Informationen unbeabsichtigt offengelegt werden. Es ist entscheidend, dass Teams Best Practices für das Geheimnismanagement anwenden, um sicherzustellen, dass vertrauliche Informationen niemals direkt im Code oder in Konfigurationsdateien hartcodiert werden. Darüber hinaus müssen Prozesse und Richtlinien etabliert werden, die den Zugriff auf Geheimnisse streng kontrollieren und ihre Verwendung in automatisierten Deployments und Pipelines sicher gestalten. Die Herausforderung besteht darin, ein Gleichgewicht zwischen Sicherheit und Benutzerfreundlichkeit zu finden, um die Produktivität nicht zu beeinträchtigen, während gleichzeitig die Sicherheit der Infrastruktur gewährleistet wird.

### Lösungsvorschläge

- **Einsatz von Secrets-Management-Tools:** Verwenden Sie dedizierte Tools wie HashiCorp Vault, AWS Secrets Manager oder Azure Key Vault, um Geheimdaten sicher zu speichern und zu verwalten. Diese Tools bieten Mechanismen, um Secrets verschlüsselt zu speichern und nur autorisierten Services und Benutzern den Zugriff darauf zu gewähren.
- **Umgebungsvariablen und CI/CD-Pipelines:** Konfigurieren Sie Ihre CI/CD-Pipelines so, dass Secrets als Umweltvariablen injiziert werden. Auf diese Weise bleiben die Secrets aus dem Code heraus und werden nur im Speicher gehalten, während die Anwendung läuft.
- **Automatisierte Rotation von Secrets:** Implementieren Sie eine Strategie zur automatischen Rotation von Secrets, um das Risiko eines unbefugten Zugriffs zu verringern. Viele Secrets-Management-Tools bieten Funktionen, um Secrets regelmäßig und automatisch zu erneuern.

# FAZIT

Infrastructure as Code revolutioniert die Art und Weise, wie Unternehmen ihre IT-Infrastruktur verwalten und betreiben. Durch die Automatisierung der Bereitstellung und den Einsatz von Code zur Verwaltung von Infrastrukturen können Unternehmen eine beispiellose Effizienz, Geschwindigkeit und Skalierbarkeit erreichen. Die Implementierung von IaC ermöglicht es, Infrastrukturänderungen schnell und konsistent durchzuführen, was zu einer verbesserten Betriebszeit und einer schnelleren Markteinführung führt.
Die Vorteile von IaC sind klar: Kostenreduktion, Risikominimierung durch menschliche Fehler und eine deutliche Beschleunigung von Entwicklungszyklen. Best Practices, wie die Verwendung von Versionskontrollsystemen, modularem Code und kontinuierlichen Tests, tragen dazu bei, diese Vorteile zu maximieren und eine robuste, sichere und wartbare Infrastruktur zu gewährleisten.

Dennoch ist der Weg zur vollständigen Adoption von IaC nicht ohne Herausforderungen. Von der Überwindung der Lernkurve innerhalb des Teams bis hin zur Handhabung von Abhängigkeiten und Secrets – jede Phase erfordert sorgfältige Planung und Fachwissen.

Nimmt man sich aber die nötige Zeit und Ressourcen, um sich tiefgehend damit auseinanderzusetzen, wird man die Entscheidung in naher Zukunft keinesfalls bereuen.

## Wie geht es jetzt weiter?

Wir bei makandra haben Erfahrung aus über 200 Projekten und eine überragende Kundenzufriedenheit. Warum? Wir arbeiten seit 14 Jahren getreu unseren Werten: Technische Exzellenz, Integrität, Förderkultur und Eigeninitiative. Davon profitieren nicht nur unser Team von Web-, DevOps-, Hosting- und UX/UI-Design-Expert*innen in Augsburg, sondern natürlich auch unsere Kund*innen. Wir beraten ehrlich und liefern nur Resultate ab, auf die wir auch stolz sein können.

Sind Sie bereit, Ihre Infrastruktur zusammen mit uns zu modernisieren?

**Kontaktieren Sie uns für eine kostenlose Erstberatung**: Wir zeigen Ihnen, wie Sie Ihre IaC-Strategie optimieren können

**Dr. Thomas Eisenbarth**
Geschäftsführer der makandra GmbH

## KONTAKTIEREN SIE UNS

+49 821 58866 180

info@makandra.de

https://makandra.de

Melli-Beese-Straße 5
86159 Augsburg

# Sie interessieren sich für eines unserer Trainings?

**Aktuell haben wir Trainings zu folgenden Themen:**

1. **Terraform 101**
   Schritt für Schritt zur Infrastrukturautomatisierung
2. **Grundlagen von Kubernetes**
   Einstieg in die Container-Orchestrierung
3. **Monitoring**
   Von der Konzeption bis zur Implementierung mit Grafana, Prometheus, Icinga oder Cloudwatch
4. **Web-Infrastrukturen bei AWS**
   Stabilität, Skalierbarkeit, Sicherheit. Bewährte Konzepte und Architekturen
5. **DevOps-Know-How für Web-Entwickler**
   Wie funktioniert IP, DNS, HTTP? Was macht ein Reverse Proxy und wofür braucht man NAT Gateways?

Schreiben Sie uns über https://makandra.de/kontakt/neu welches Training Sie gerne durchführen würden oder bei Fragen zu den Trainings.