GitLab

GitOps-Leitfaden für Anfänger(innen)

- Einführung in GitOps
- Die Vorteile der Infrastruktur-Automatisierung
- Best Practices für GitOps

Inhalt

/03/ **Einführung**

/04/ **Der Weg zu GitOps**

/06/ **So funktioniert GitOps**

/08/ **Die Vorteile von GitOps**

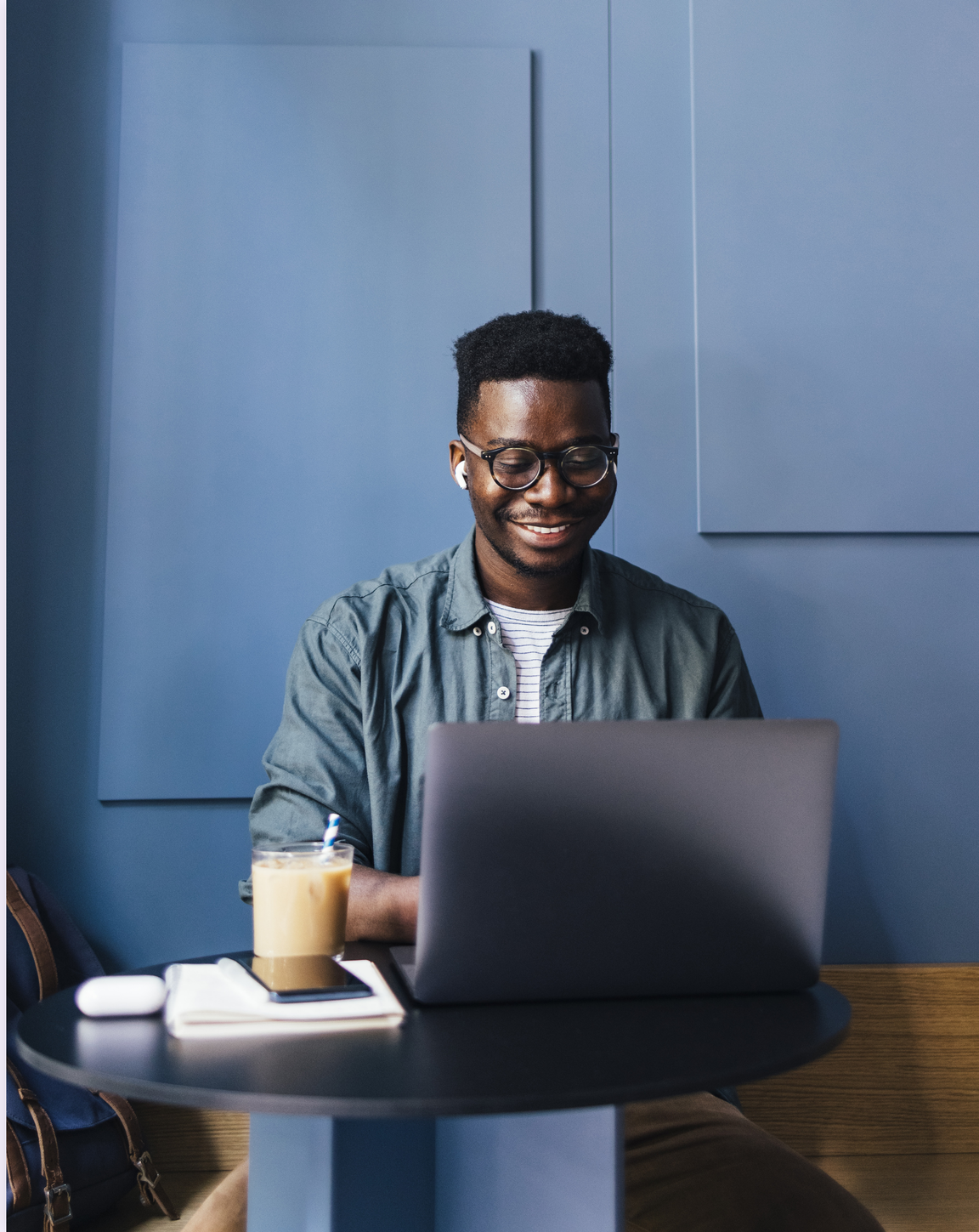
/09/ **Gängige GitOps-Tools**

/10/ **Best Practices für den Einstieg in GitOps**

- Definiere die gesamte Infrastruktur als Konfigurationsdateien
- Dokumentiere, was du nicht automatisieren kannst
- Entwickle einen Prozess für Code Reviews und Merge Requests
- Berücksichtige mehrere Umgebungen
- Mache CI/CD zum Zugriffspunkt auf Ressourcen
- Nutze eine Repository-Strategie
- Mache nur kleine Änderungen
- CI/CD-Pipelines mit GitOps und Terraform

/15/ **Die Zukunft der Infrastrukturautomatisierung**

/16/ **Über GitLab**



Einführung

Softwareanwendungen werden immer ausgefeilter, wodurch auch die Anforderungen an die Infrastruktur steigen. Infrastruktur-Teams müssen komplexe Bereitstellungen in einem riesigen Maßstab und noch dazu in enormer Geschwindigkeit ermöglichen.

Ähnlich wie die Anwendungsentwicklung wird auch die Infrastruktur zunehmend automatisiert, wobei wiederholbare und zuverlässige Möglichkeiten zur Bereitstellung von Softwareumgebungen erwartet werden.

Infrastructure as Code (IaC) und GitOps sind zu modernen Ansätzen geworden, um komplexe Bereitstellungen in großem Maßstab zu unterstützen.

In diesem E-Book lernst du die Infrastruktur-Automatisierungsprozesse von GitOps kennen und erfährst, „inwiefern es eine Komplettlösung“ für die Entwicklung, Änderung und Bereitstellung von Infrastruktur darstellt. In diesem E-Book erfährst du außerdem:

- Wie GitOps mit Prozessen funktioniert, die du bereits in der Anwendungsentwicklung nutzt
- Welche drei Komponenten für deine Teams nötig sind, um mit GitOps durchzustarten
- Welche Best Practices und Workflows es für GitOps gibt

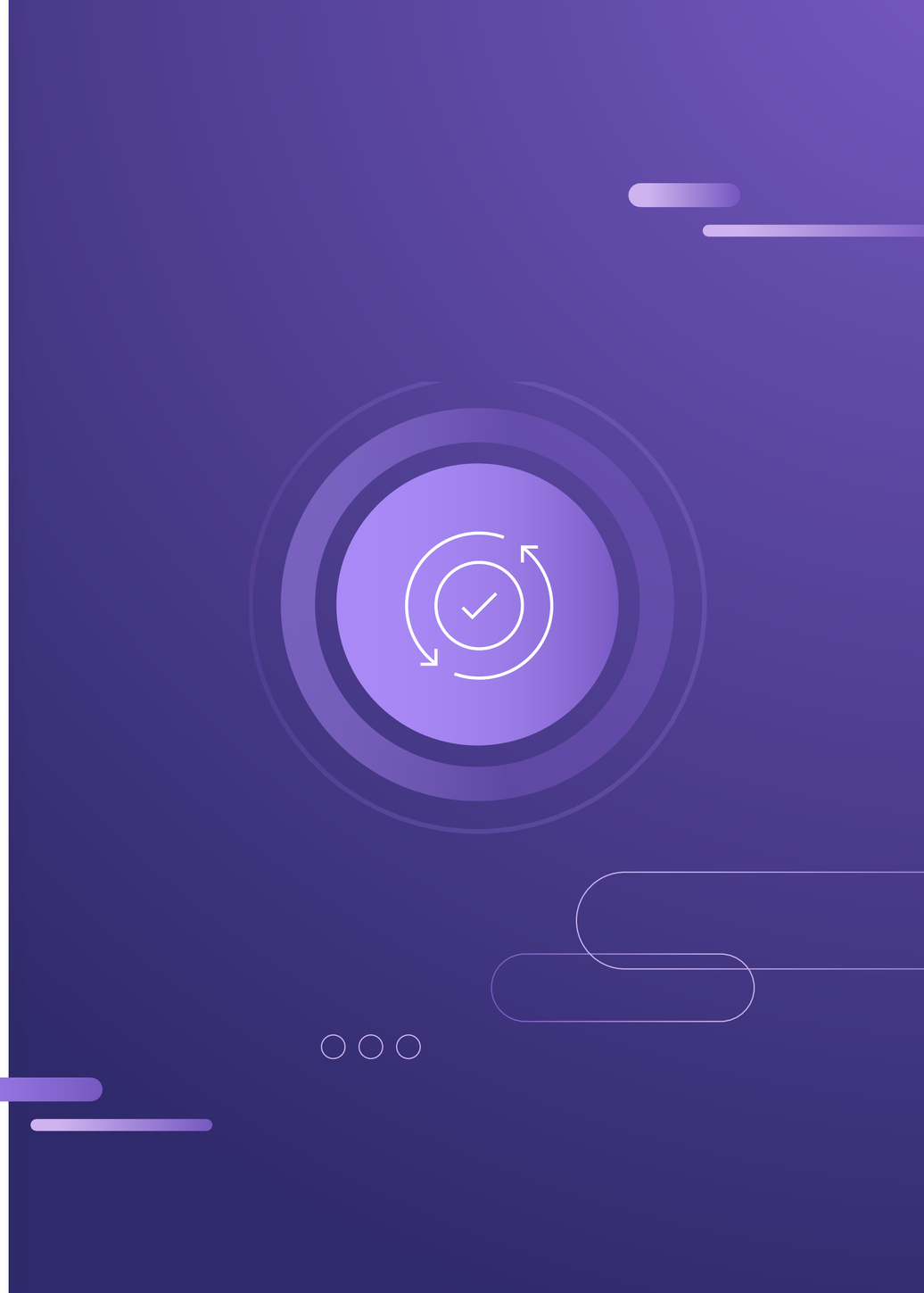
Der Weg zu GitOps

AWS ist seit 2006 öffentlich verfügbar, doch auch schon zuvor war das lokale Infrastruktur-management eine mühselige Aufgabe für IT-Teams. Auf verschiedenen Servern wurden unterschiedliche Anwendungen und Dienste ausgeführt, und bei einer Skalierung musste die IT einen ganzen Server einrichten und die gleichen Anwendungen mit den gleichen Einstellungen erneut installieren. Zum Glück wurden Tools entwickelt, die das etwas vereinfachten.

Die erste Generation von Konfigurationsmanagement-Tools (CM-Tools) wie **Puppet** und **Chef** erleichterten die Einrichtung bestehender Server. Die IT konnte nun einen Server oder eine VM hochfahren, den Puppet- oder Chef-Agent installieren und dann das Tool arbeiten lassen, denn es richtete alles ein, was zum Ausführen der Anwendungen auf dem Server nötig war. Diese Tools liefen auf lokalen Servern und auf Cloud-Servern.

CM-Tools der ersten Generation waren eine effiziente Möglichkeit, die manuellen Schritte einfach zu replizieren, die bei der Einrichtung eines neuen Produktionsservers nötig waren. Da diese Schritte nun automatisiert waren, wurde die Einrichtung von neuen Servern um ein Vielfaches einfacher. Trotzdem war es noch immer nicht möglich, neue VMs bereitzustellen, und auch die Cloud-native Infrastruktur bereitete Probleme.

Dann kamen CM-Tools der zweiten Generation auf den Markt, wie **Ansible** und **SaltStack**. Diese Tools können – ähnlich wie CM-Tools der ersten Generation – Software auf einzelnen Servern installieren, sie können aber zusätzlich auch VMs bereitstellen und diese anschließend einrichten. Beispielsweise können sie zehn EC2-Instanzen erstellen und dann die benötigte Software auf all diesen Instanzen installieren.



Ein bedeutender Nachteil dieser CM-Tools ist jedoch, dass sie nur Server und VMs bereitstellen und einrichten können. Sie sind nicht für Cloud-native Dienste geeignet.

Amazon CloudFormation erschien etwa zeitgleich mit den CM-Tools der zweiten Generation. Es übernimmt nicht die Servereinrichtung, bietet Anwender(inne)n dafür aber die Möglichkeit, deklarativen Code zu nutzen, um die gesamte AWS-Anwendungsarchitektur bereitzustellen. So musst du dich nicht mehr durch die Verwaltungskonsole klicken, um Ressourcen manuell zu erstellen. Du kannst deine Infrastruktur also einfach als JSON oder YAML beschreiben und sie über die **AWS**-Verwaltungskonsole, die Befehlszeilenschnittstelle (CLI) oder das AWS SDK bereitstellen. Da es ein Amazon-Dienst ist, funktioniert das jedoch nur für AWS.

Microsoft Azure bietet mit dem Azure Resource Manager (ARM) ein ähnliches Tool, mit dem das du deine Infrastruktur in JSON-Vorlagen beschreiben kannst. Ähnlich wie Amazon CloudFormation nur für AWS funktioniert, kann ARM nur für Azure-Dienste eingesetzt werden.

Als private Clouds und andere öffentliche Clouds wie Azure und **Google Cloud** immer beliebter wurden, wechselten viele Unternehmen zu einer anderen Cloud oder setzten mehrere Clouds ein, um nicht von einer Cloud-Plattform abhängig zu sein. Um diesen Veränderungen gerecht zu werden, erschienen **Multicloud**-CM-Tools wie **Terraform und OpenTofu**. Beschreibe einfach deine Dienste und stelle sie für mehrere Clouds/Anbieter/Cloud-Dienste bereit.

Ein Vorteil dieser Tools ist es, dass sie tolle Möglichkeiten für Versionskontrolle, Code Reviews und **kontinuierliche Integration/kontinuierliche Lieferung (CI/CD)** für Infrastrukturcode bieten.

So funktioniert GitOps

GitOps setzt auf bewährte DevOps-Prozesse und wendet sie auf Infrastrukturcode an. Wie der Name vermuten lässt, kombiniert es „Git“ und „Operations“, also das Ressourcenmanagement. Wie bei DevOps hat GitOps das Ziel, mithilfe von CI/CD deine Ressourcen automatisch bereitzustellen, indem Code verwendet wird, der in deinen Git-Repositorys gespeichert ist.

Mit GitOps wird dein Infrastrukturdefinitionscode als JSON oder YAML definiert und in einem .git-Ordner in einem Projekt gespeichert. Er liegt also in einem Git-Repository, das als einzige Quelle der Wahrheit dient. Wenn du Git für die Infrastruktur verwendest, kannst du Folgendes tun:

- **Zeige den vollständigen Änderungsverlauf für den Infrastrukturcode deines Unternehmens an.** Teams können bei Bedarf auf frühere Versionen zurückgreifen.
- **Führe Code Reviews für deine Infrastruktur durch.** Der Code Review ist ein wichtiger Aspekt von DevOps und stellt sicher, dass fehlerhafter Anwendungscode nicht in die Produktion gelangt. Dies ist auch für Infrastrukturcode wichtig. Ein fehlerhafter Infrastrukturcode kann eine kostspielige Cloud-Infrastruktur unabsichtlich aufblähen und das Unternehmen so tausende Euro pro Stunde kosten. Ebenso kann ein fehlerhaftes Skript dazu führen, dass deine Anwendungen nicht ausgeführt werden können und dein Dienst dadurch ausfällt. Durch Code Reviews lassen sich solche Fehler vermeiden, indem mehrere Personen jede Änderung sehen, bevor sie freigegeben wird.
- **Nutze kontinuierliche Integration und Bereitstellung.** Mit Tools wie GitLab CI/CD kannst du (aktualisierten) Infrastrukturode automatisch bereitstellen und ihn automatisch auf deine Cloud-Umgebung anwenden. Ressourcen, die zum Infrastrukturcode hinzugefügt werden, werden automatisch bereitgestellt und nutzbar gemacht. Ressourcen, die geändert wurden, werden in deiner Cloud-Umgebung aktualisiert, während Ressourcen, die aus dem Infrastrukturcode entfernt wurden, automatisch entfernt und gelöscht werden. Auf diese Weise kannst du Code schreiben, ihn per Commit in dein Git-Repository übernehmen und alle Vorteile von DevOps nutzen – für deine Infrastruktur.

GitOps = IaC + MRs + CI/CD

- **IaC** – GitOps nutzt ein Git-Repository als einzige Quelle der Wahrheit für Infrastrukturdefinitionen. Ein Git-Repository ist ein .git-Ordner in einem Projekt, in dem alle Änderungen an Dateien eines Projekts erfasst werden. Infrastructrue as Code (IaC) bedeutet, dass alle Infrastrukturkonfigurationen als Code gespeichert werden. Der tatsächliche gewünschte Zustand ist möglicherweise als Code (z. B. Anzahl von Repliken oder Pods) gespeichert.
- **MRs** – GitOps nutzt Merge Requests (MRs) als Änderungsmechanismen für alle Infrastruktur-Updates. Im MR können Teams über Reviews und Kommentare zusammenarbeiten und dort finden formelle Approvals statt. Ein Merge wirkt sich per Commit auf den main-Branch (oder Trunk) aus und dient als Änderungsprotokoll für Audits und die Problembehandlung.
- **CI/CD** – GitOps automatisiert Infrastruktur-Updates über einen Git-Workflow mit kontinuierlicher Integration und Lieferung (CI/CD). Wenn neuer Code zusammengeführt wird, führt die CI/CD-Pipeline die Änderung in der Umgebung durch. Jede Konfigurationsverschiebung, z. B. manuelle Änderungen oder Fehler, wird von der GitOps-Automatisierung überschrieben, sodass die Umgebung dem in Git definierten gewünschten Zustand entspricht. GitLab verwendet CI/CD-Pipelines, um die GitOps-Automatisierung zu verwalten und zu implementieren. Es können aber auch andere Formen der Automatisierung, wie z. B. Definitionsoperatoren, verwendet werden.

Die Vorteile von GitOps

Ein GitOps-Framework ermöglicht die Automatisierung der Infrastruktur. Diese Automatisierung selbst ist schon ein Vorteil, sie ist aber nicht der einzige positive Aspekt von GitOps. Unternehmen, die GitOps einsetzen, profitieren von weiteren langfristigen Vorteilen.

Zusammenarbeit bei Infrastrukturänderungen. Da jede Änderung den gleichen Prozess für Änderungen, Merge Requests, Überprüfungen und Genehmigungen durchläuft, können sich die leitenden Entwickler(innen) auf andere Bereiche als das kritische Infrastrukturmanagement konzentrieren.

Schnellere Markteinführung. Die Ausführung per Code ist schneller als manuelle Arbeiten. Testfälle werden automatisiert und sind wiederholbar, sodass schnell stabile Umgebungen bereitgestellt werden können.

Vereinfachtes Auditing. Wenn Infrastrukturänderungen manuell über verschiedene Interfaces ausgeführt werden, kann dies das Auditing komplex und zeitaufwändig machen. Für das Audit müssen dann nämlich Daten von verschiedenen Orten abgerufen und vereinheitlicht werden. Mit GitOps werden alle an Umgebungen vorgenommenen Änderungen im Git-Log gespeichert, was den Auditprozess vereinfacht.

Geringeres Risiko. Alle Änderungen an der Infrastruktur werden über Merge Requests nachverfolgt, sodass sie einfach auf einen früheren Zeitpunkt zurückgesetzt werden können.

Geringere Fehleranfälligkeit. Die Infrastrukturdefinition ist kodifiziert und wiederholbar, wodurch sie weniger anfällig für menschliche Fehler ist. Dank Code Reviews und Zusammenarbeit bei Merge Requests werden Fehler erkannt und behoben, bevor sie überhaupt in die Produktion gelangen.

Geringere Kosten und Ausfallzeiten. Durch die Automatisierung der Infrastrukturdefinition und -tests werden manuelle Arbeiten reduziert, die Produktivität verbessert und Ausfallzeiten durch die integrierte Wiederherstellungs-/Zurücksetzungsfunktion reduziert. Die Automatisierung ermöglicht es den Infrastrukturteams auch, ihre Cloud-Ressourcen besser zu verwalten, was zudem die Cloud-Kosten optimieren kann.

Verbesserte Zugriffskontrolle. Es ist nicht notwendig, allen Infrastrukturkomponenten Zugriffsrechte zu geben, da Änderungen automatisiert sind (nur CI/CD benötigt Zugriff).

Zusammenarbeit für Compliance. In stark regulierten Kontexten schreiben Richtlinien oft vor, dass so wenig Personen wie möglich Änderungen an einer Produktumgebung vornehmen dürfen. Mit GitOps kann fast jedes Teammitglied Änderungen über Merge Requests vorschlagen, sodass ein höchst kollaboratives Umfeld entsteht und gleichzeitig die Anzahl der Personen eingeschränkt wird, die den Production-Branch zusammenführen dürfen. So wird die Compliance sichergestellt.

Gängige GitOps-Tools

GitOps zeichnet sich durch einen Aspekt aus: Es ist nämlich kein einzelnes Produkt, kein einzelnes Plugin und keine einzelne Plattform. Viel mehr ist GitOps ein Framework, das Teams dabei unterstützt, ihre IT-Infrastruktur durch Prozesse zu verwalten, die bereits bei der Anwendungsentwicklung eingesetzt werden. Beliebte Tools sind Ansible, Terraform und Kubernetes, doch der GitOps-Prozess ist grundsätzlich weitgehend technologieunabhängig (natürlich mit Ausnahme von Git).

GitOps eignet sich für eine Vielzahl von Szenarien. GitOps und Kubernetes passen beispielsweise besonders gut zusammen. **Kubernetes** ist für alle großen Cloud-Plattformen geeignet und nutzt zustandslose und unveränderliche Container. Da containerbasierte Apps, die in Kubernetes ausgeführt werden, eigenständig sind, musst du nicht für jede App Server bereitstellen und konfigurieren. Stelle Kubernetes-Cluster und andere benötigte Infrastruktur wie Datenbanken und Netzwerke mit Terraform bereit.

Bei der Bereitstellung zustandsabhängiger Anwendungen sind zusätzliche Überlegungen nötig, um Daten auf externen Diensten, wie etwa in einer Amazon-Aurora-Datenbankinstanz oder in einem Redis-Cache zu speichern. Kubernetes eignet sich zwar grundsätzlich gut für ein GitOps-Framework, ist aber keine Voraussetzung für GitOps. Du kannst es auch mit traditionellen Cloud-Infrastrukturen wie VMs nutzen. In diesem Fall stellst du mit Terraform bereit und nutzt dann ein CM-Tool wie Ansible, um neue VMs zu konfigurieren.

Git-Code-Repository	git		
Git-Verwaltungstool	GitLab	GitHub	Bitbucket
Tool für kontinuierliche Integration	GitLab	Jenkins	circleci
Tool für kontinuierliche Lieferung	GitLab	Spinnaker	flux
Container-Registry	GitLab	docker hub	
Konfigurationsmanager	ANSIBLE	CHEF	puppet
Infrastrukturbereitstellung	Terraform	OpenTofu	AWS CloudFormation
Container-Orchestrierung	kubernetes		

Best Practices für den Einstieg in GitOps

Definiere die gesamte Infrastruktur als Konfigurationsdateien

Beschreibe deine gesamte Infrastruktur mit deklarativen IaC-Konfigurationsdateien, die den gewünschten Endzustand angeben, anstatt Schritt-für-Schritt-Anweisungen zu geben. Um die besten Ergebnisse zu erzielen, solltest du Tools wählen, die für eine deklarative Syntax entwickelt wurden, sowie deine gesamte Infrastruktur einbeziehen – einfache Dienste mit Standardeinstellungen. Manuelle Ausnahmen verursachen Technical Debt und untergraben deine GitOps-Strategie.

Wenn du IaC bereits nutzt, füge deinen Infrastrukturcode zu deinem GitOps-Repository hinzu. Wenn nicht, konvertiere vorhandene Ressourcen mit Tools wie AWS CloudFormation Export oder Terraform Import. Versuche nicht, deine gesamte Infrastruktur auf einmal zu automatisieren, denn das könnte eine überwältigende Mammutaufgabe sein. Arbeite stattdessen iterativ, um nach und nach mehr Infrastruktur in GitOps-Workflows zu verschieben.

Dokumentiere, was du nicht automatisieren kannst

Es ist nicht immer möglich, alles zu automatisieren. Azure hat beispielsweise einige (normalerweise neuere) Einstellungen, die noch nicht zu ARM-Vorlagen hinzugefügt wurden. Eine gängige Problemumgehung ist die Nutzung von PowerShell.

Ein weiteres Beispiel ist die Zusammenarbeit mit Drittanbietern. Stell dir vor, du arbeitest mit einem Anbieter zusammen, der deine IP-Adressen manuell auf die Genehmigungsliste setzen muss. Eine Anfrage für eine Aufnahme in die Genehmigungsliste kann nur von einem Manager bzw. einer Managerin gestellt werden. Also muss für jeden neuen Dienst und jede neue Umgebung die IP-Adresse manuell herausgesucht, an deine(n) Manager(in) weitergegeben und dann per E-Mail an den Anbieter gesendet werden. Stelle sicher, dass solche Prozesse sehr gut dokumentiert sind.

Aller Wahrscheinlichkeit nach wirst du immer über einige Legacy-Umgebungen verfügen, die manuell bearbeitet werden müssen. Dokumentiere diese Instanzen, damit sie nicht vergessen werden.

Für Teams, die oft kleine, manuelle Änderungen an der Infrastruktur vornehmen, kann die Einführung eines Prozesses wie GitOps eine große Umstellung bedeuten. GitOps ist ein Framework, bei dem Infrastrukturteams neue Gewohnheiten annehmen und alte Gewohnheiten hinter sich lassen müssen. Dies kann einige Zeit dauern und liegt nicht jedem Team gleichermaßen. Best Practices, auf die häufig verwiesen wird, sind hilfreich für eine langfristig erfolgreiche GitOps-Strategie.

Entwickle einen Prozess für Code Reviews und Merge Requests

Es ist wichtig, GitOps-Teams mit Git und Code Reviews vertraut zu machen. Manche Teams nutzen bereits ein Git-Repository als Speicherort für Konfigurationscode, verwenden aber gewisse Funktionen wie Merge Requests noch nicht. Sieh dir zunächst die Richtlinien für Code Reviews des **Open-Source-Projekts von GitLab (in englischer Sprache verfügbar)** an. Dieses Projekt gibt dir ein Gefühl dafür, welche Arten von Informationen du schlussendlich in deine Code-Review-Richtlinien aufnehmen möchtest.

Lege eine **Mindestanzahl an Prüfer(inne)n (in englischer Sprache verfügbar)** fest, sodass Code immer von einer gewissen Mindestanzahl an Teammitgliedern überprüft wird, bevor ein Merge Request genehmigt wird.

Für Teams, die GitOps erstmalig nutzen, besteht auch die Möglichkeit, „optionale Reviews“ anstelle von „erforderlichen blockierenden Reviews“ einzurichten. Da dies ein neuer Prozess ist, solltest du dir entsprechend Zeit dafür nehmen, dich mit Code Reviews vertraut zu machen und einen guten Rhythmus zu entwickeln. Sobald die Teams mit dem Toolset und den Praktiken vertraut sind, kannst du verpflichtende Reviews einführen, um sicherzustellen, dass der Code jedes Mal geprüft wird.

Wie wir schon an anderer Stelle empfohlen haben – fange klein und einfach an. Wenn du schon zu Beginn ein komplexes Set aus Richtlinien einsetzt, wird niemand deinen Prozess nutzen wollen. Konzentriere dich eher darauf, dass die Neuerungen akzeptiert werden, anstatt dich strikt an die beste Vorgehensweise zu halten. Im Laufe der Zeit kannst du Code-Review-Richtlinien nach und nach iterativ einführen, damit sie robust und vollständig sind.

Berücksichtige mehrere Umgebungen

Es ist empfehlenswert, über mehrere Umgebungen zu verfügen. Ein gutes Beispiel sind DTAP-Umgebungen: Entwicklung (D für Development), Test (T), Akzeptanz (A) und Produktion (P). Code kann an die Entwicklungs- oder Testumgebung ausgerollt werden, sodass du anschließend testen kannst, ob die Dienste weiterhin verfügbar sind und wie erwartet funktionieren. Wenn ja, kannst du die Änderungen für die nächsten Umgebungen ausrollen.

Nachdem du deinen Code für deine Umgebungen ausgerollt hast, ist es wichtig, ihn mit deinen laufenden Diensten synchron zu halten. Sobald du weißt, dass es einen Unterschied zwischen deinem System und einer Konfiguration gibt, kannst du einen der beiden Punkte anpassen. Eine Lösung für dieses Problem besteht darin, unveränderliche Images wie etwa Container zu verwenden, damit weniger Unterschiede auftreten können.

Tools wie **Chef**, **Puppet** und **Ansible** verfügen über Funktionen wie „diff alert“, wobei du eine Warnung erhältst, wenn sich deine Dienste von deinem Konfigurationscode unterscheiden. Mit Tools wie **Kubediff** und **Terradiff** kannst du diese Funktion auch für Kubernetes und Terraform nutzen.

Mache CI/CD zum Zugriffspunkt auf Ressourcen

Wenn du deine CI/CD-Tools zum Zugriffspunkt für Cloud-Ressourcen machst, erleichterst du die Nutzung eines GitOps-Workflows und reduzierst manuelle Änderungen an der Cloud-Infrastruktur.

Natürlich kann es in der frühen Entwicklungsphase für Teams hilfreich zu sein, diesen Zugriff zu haben und ihren Code zu schreiben. Außerdem gibt es verschiedenste Gründe, warum du gelegentlich Zugriff benötigst. Es hilft dir jedoch, den GitOps-Prozess einzuführen und einzuhalten, wenn du dich eher darauf konzentrierst, warum du Zugriff brauchst, anstatt dich auf Ausnahmen zu konzentrieren, warum du keinen Zugriff haben solltest.

Nutze eine Repository-Strategie

Überlege dir, wie du deine Repositorys einrichten möchtest. Vielleicht möchtest du ein einziges Repository für deine gesamte Infrastruktur nutzen. Es ist sinnvoll, ein Repository für deine gemeinsam genutzten Ressourcen zu haben, aber den Code für dienstspezifische Ressourcen dieser Dienste zu behalten. Ein anderer Ansatz ist es, Ressourcen für verschiedene Projekte in unterschiedlichen Repositorys zu speichern. Dies hängt von der Struktur deines Unternehmens, deinen Diensten und deinen persönlichen Präferenzen ab.

Ein paar Aspekte solltest du bei der Entscheidung für eine Mono- oder Multi-Repository-Strategie beachten:

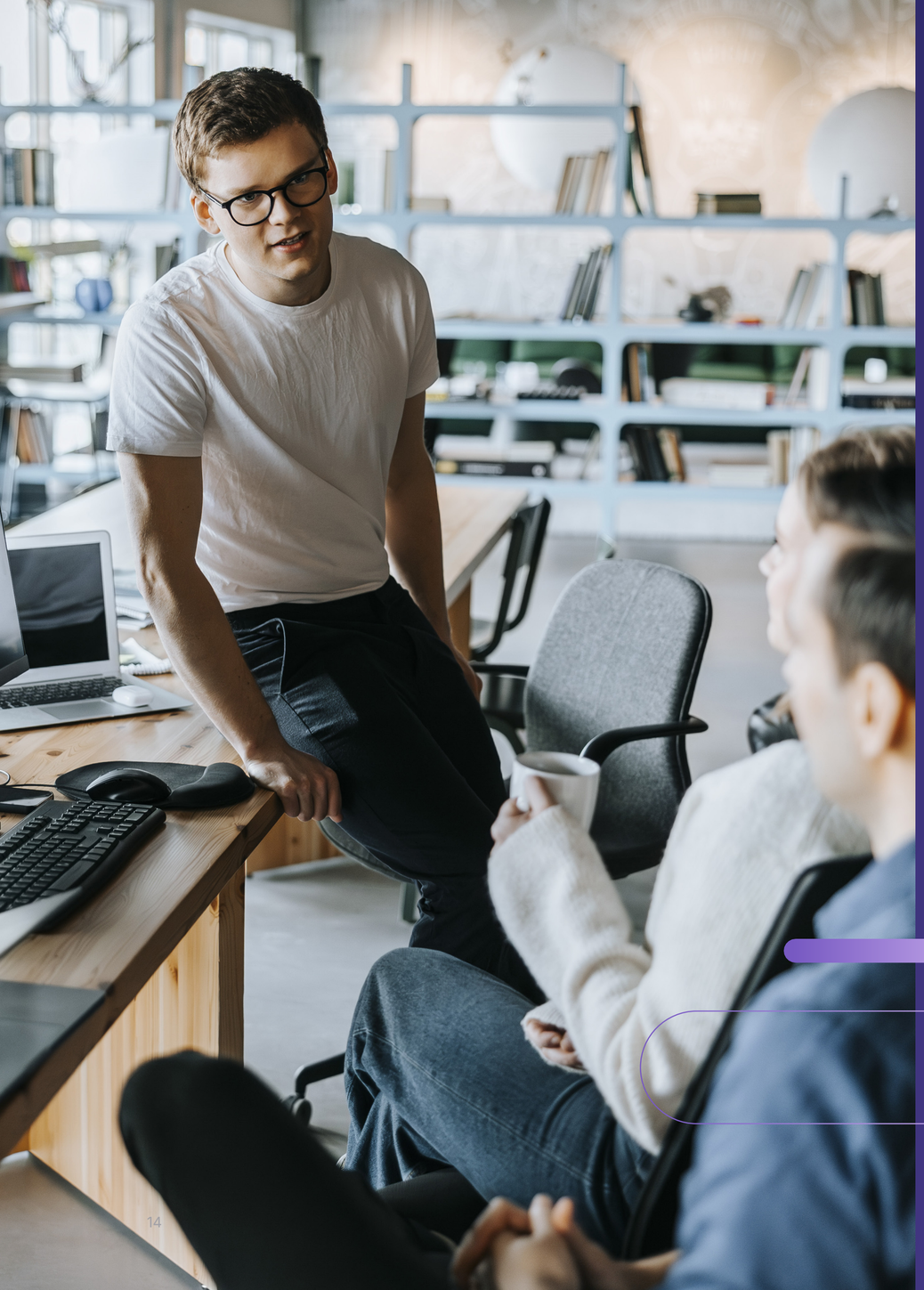
- Hast du häufig Auftragnehmer(innen) oder Personen, die an Projekten arbeiten, bei denen es ein Risiko sein könnte, wenn sie Zugriff auf den gesamten Code haben?
- Musst du mehrere Abhängigkeiten berücksichtigen?
- Möchtest du, dass dein Repository als einzige Quelle der Wahrheit dient?

Google, eines der größten Technikunternehmen der Welt, **nutzt ein einziges Repository für seinen gesamten Code (in englischer Sprache verfügbar)**. HashiCorp empfiehlt, dass jedes Repository, das Terraform-Code enthält, **ein verwaltbarer Infrastrukturteil (in englischer Sprache verfügbar)** ist, wie etwa eine Anwendung, ein Dienst oder eine bestimmte Art von Infrastruktur (wie eine allgemeine Netzwerkinfrastruktur). Die Definition von „verwaltbar“ kann natürlich variieren.

Ziehe auch eine Git-Branching-Strategie in Betracht, wie zum Beispiel **Feature-Branching (in englischer Sprache verfügbar)**, sodass mehrere Personen zeitgleich am selben Repository arbeiten können.

Mache nur kleine Änderungen

Egal, was du tust: Achte darauf, dass deine Commits klein sind. Dadurch erhältst du ein engmaschiges Änderungsprotokoll, mit dem du unkompliziert ein Rollback der einzelnen Änderungen vornehmen kannst. Dieser Prozess wird bei GitLab als **Iteration (in englischer Sprache verfügbar)** bezeichnet und ist der Grund, warum wir Änderungen schnell vornehmen können. Durch kleinere Schritte und die Bereitstellung kleinerer, einfacherer Funktionen erhalten wir schneller ein Feedback.



CI/CD-Pipelines mit GitOps und Terraform

Richte deine CI/CD-Pipeline so ein, dass zuerst Infrastrukturcode validiert wird, beispielsweise mit dem Terraform-Befehl **validate** oder mit einem Linter für JSON-Dateien. Dein Infrastrukturcode sollte so behandelt werden, als wäre er Produktionscode. Dein Produktionscode sollte clean und konsistent sein.

Wenn jemand ungültigen Code committet, stelle sicher, dass das Build oder die Validierung fehlschlägt und das Team sofort benachrichtigt wird. So kann das Problem schnell durch einen Fix oder ein Rollback des Commit behoben werden. Wenn du kleine Änderungen vornimmst, wird es auch einfacher, Probleme zu finden.

Wenn der Code gültig ist, sollte die CI/CD-Pipeline alle Befehle ausführen, die zum Bereitstellen der im Konfigurationscode definierten Infrastruktur nötig sind. Beispielsweise den Terraform-Befehl **apply** oder **AWS update-stack** für CloudFormation.

Ein Beispiel für ein GitOps-Projekt, bei dem Terraform, CI/CD und Kubernetes verwendet werden, findest du in unserer Gruppe **gitOps-demo**. Dort erwarten dich Links zu Terraform-Sicherheitsempfehlungen, Terraform-Code, der eine Konfiguration aller drei großen Cloud-Anbieter repräsentiert, sowie Anweisungen für die Reproduktion dieser Demo in deiner eigenen Gruppe.

Zur Gruppe gitops-demo >

Die Zukunft der Infrastrukturautomatisierung

GitOps ist keine Magie: Es nutzt lediglich bereits bekannte IaC-Ops-Tools und bindet sie in einem Workflow ein, der an DevOps angelehnt ist. Dadurch werden die Versionsverfolgung verbessert, kostspielige Fehler reduziert und eine schnelle, automatisierte Infrastrukturbereitstellung ermöglicht, die in einem **Setup mit mehreren Umgebungen und sogar in einem Multicloud-Setup (in englischer Sprache verfügbar)** wiederholt werden kann.

GitLab hilft dir bei den ersten Schritten mit einem GitOps-Workflow. Mit GitLab kannst du physische, virtuelle und Cloud-native Infrastrukturen (darunter auch Kubernetes und Serverless-Technologien) verwalten. Außerdem bietet GitLab enge Integrationen mit verschiedenen Tools für die Infrastrukturautomatisierung wie Terraform, OpenTofu, AWS CloudFormation, Ansible, Kubernetes, Docker, Chef, Puppet und anderen an. Zusätzlich zu einem Git-Repository warten mit GitLabCI/CD, Merge Requests, agile Planung, Sicherheitsscans, Richtlinienverwaltung, verschiedene KI-basierte Funktionen und ein einfaches Single Sign-On auf dich, sodass alle im Team von einer Plattform aus zusammenarbeiten und für alle Cloud-Anbieter bereitstellen können.

Über GitLab

GitLab ist die umfassendste, KI-gestützte DevSecOps-Plattform für Softwareinnovationen. GitLab bietet eine Schnittstelle, einen Datenspeicher, ein einheitliches Berechtigungsmodell, einen Wertstrom, eine Serie von Berichten, einen Ort zum Sichern deines Codes, einen Ort für die Bereitstellung in jeder Cloud und einen Ort, an dem jede und jeder einen Beitrag leisten kann. Die Plattform ist die einzige echte Cloud-agnostische, durchgängige DevSecOps-Plattform, die alle DevSecOps-Funktionen an einem Ort vereint.

Mit GitLab können Unternehmen Code schnell und kontinuierlich erstellen, bereitstellen und verwalten, um ihre Geschäftsvision in die Realität umzusetzen. GitLab ermöglicht Kund(inn)en und Anwender(inne)n schnellere Innovationen, eine einfachere Skalierung sowie eine effektivere Betreuung und Bindung von Kund(inn)en. Als Open-Source-Lösung agiert GitLab an der Seite seiner wachsenden Community aus Tausenden Entwickler(inne)n und Millionen Anwender(inne)n, um kontinuierlich neue Innovationen zu liefern.

Wenn du wissen möchtest, wie wir dir bei deinen ersten Schritten mit GitOps helfen können, kannst du GitLab kostenlos testen.

Starte mit deiner kostenlosen GitLab-Testversion >

GitLab