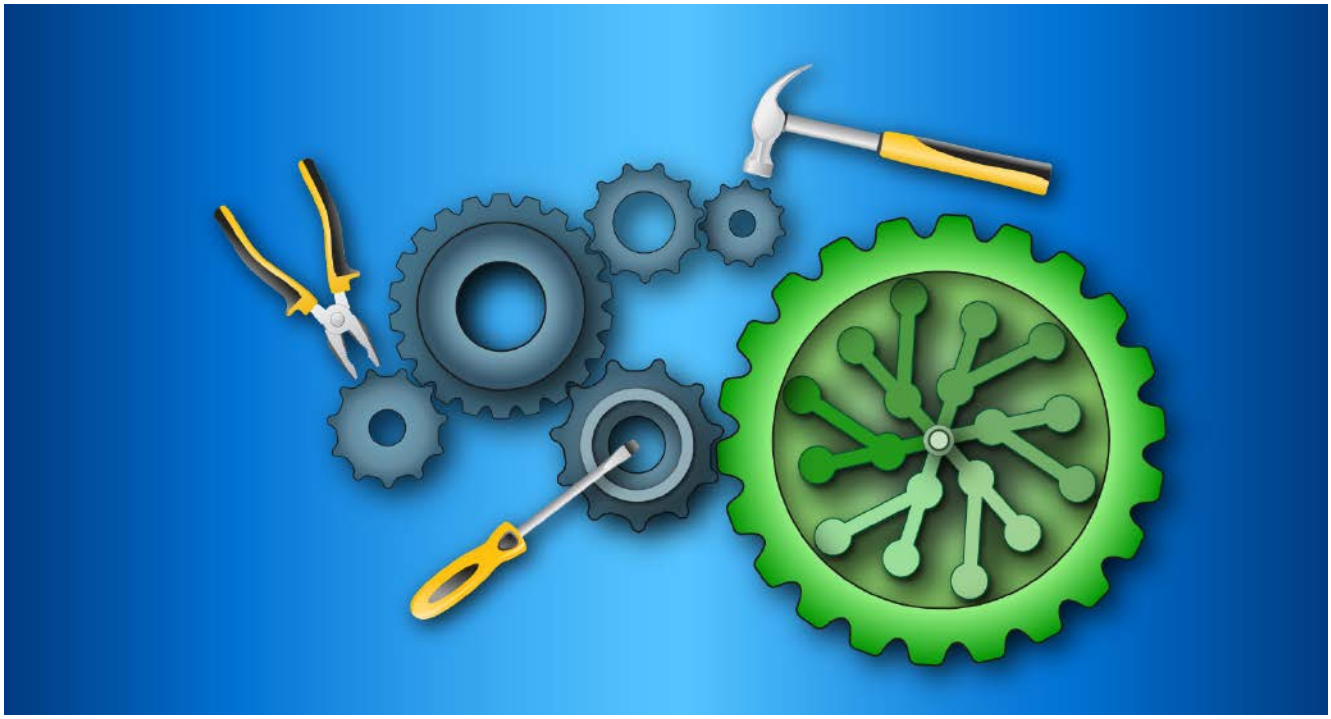


Marktübersicht: GitOps-Werkzeuge im Vergleich

Automatisierungsgehilfen

Philipp Markiewka, Johannes Schnatterer

Wer von klassischen CI/CD-Umgebungen auf GitOps umsteigen will, muss sich für eines von zahlreichen Werkzeugen entscheiden. Doch nicht immer erschließen sich Funktionsumfang und Eignung auf den ersten Blick. Eine Bestimmungshilfe.

Der Begriff GitOps setzt sich zusammen aus dem Namen der Versionsverwaltung Git und der Abkürzung Ops für Operations, den IT-Betrieb. Die Idee zu diesem zusätzlichen Tool im DevOps-Werkzeugkasten entstammt dem Kubernetes-Umfeld und verspricht eine neue Stufe der IT-Automatisierung. Wie der Continuous-Delivery-Ansatz setzt GitOps darauf, alle Informationen in der Sourcecodeverwaltung zu pflegen. Der Unterschied ist jedoch, dass die Deployment-Umgebung ihren Zustand direkt aus Git synchronisiert und nicht der CI-Server das Ausrollen übernimmt. Die Konfiguration muss im Git also versioniert sein. Da diese wie Code behandelt wird, spricht man von „Infrastructure as Code".

Kein Wunder, dass mittlerweile eine wachsende Zahl an GitOps-Werkzeugen zur Wahl steht. Doch welchen Funktionsumfang bieten diese, reicht ein einziges aus und ist damit „alles" automatisierbar? Diese und ähnliche Fragen beantwortet dieser Artikel anhand konkreter Beispiele. Er erarbeitet Auswahlkriterien und illustriert diese durch einen Vergleich der bekannten GitOps-Werkzeuge ArgoCD und Flux v2.

Unübersichtlicher Markt

Eine Übersicht über die am Markt verfügbaren GitOps-Werkzeuge ist nicht so trivial, wie es zunächst klingt. Das liegt zum einen daran, dass um den Begriff ein gewisser Hype besteht und Produktanbieter sich sehr gerne mit der Bezeichnung GitOps schmücken. Zum anderen ist es schwer, den Begriff eindeutig zu definieren, und er kommt auf unterschiedlichen Ebenen des Stacks (von der physischen Infrastruktur bis zur laufenden Anwendung in der Cloud) mit unterschiedlichem Reifegrad zum Einsatz. Der Artikel „Finger weg" widmet sich dem Thema ausführlicher (siehe S. 42, [1]).

Websites wie die von Weaveworks gestartete awesome-gitops oder die von INNOQ-Mitarbeitern bereitgestellte gitops.tech geben einen ersten Überblick über verfügbare Tools (siehe ix.de/zc9k). Wie ein genauerer Blick zeigt, erfüllen die gelisteten Werkzeuge unterschiedlichste Aufgaben bei der Implementierung von GitOps und natürlich heben sie sich auch in Verbreitung, Reifegrad und Aktivität voneinander ab. Aus den diversen Anwendungsfällen extrahiert dieser Artikel drei Kategorien: Werkzeuge für Kubernetes, ergänzende Werkzeuge und infrastrukturnahe Werkzeuge. Zusätzlich fasst je-

weils eine Tabelle die Werkzeuge und ihre Eigenschaften zusammen. Für eine bessere Einschätzung von Verbreitung, Reifegrad und Aktivität enthalten die Tabellen außerdem verschiedene Git- und GitHub-basierte Metriken (Stand Februar 2021).

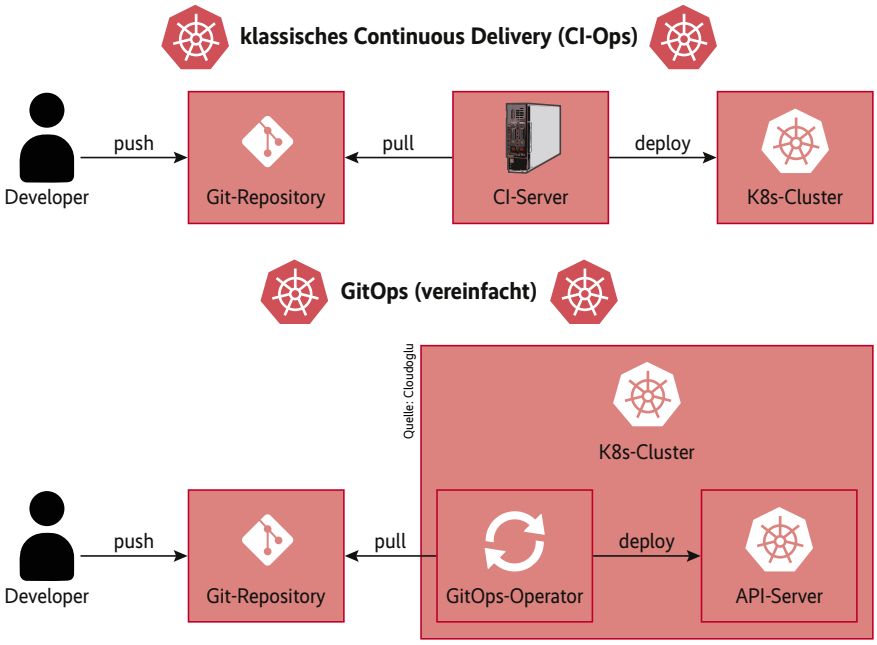
Werkzeuge für Kubernetes

Bei GitOps-Werkzeugen kommt als Erstes meist das Thema Operatoren für Kubernetes auf. Generell ist ein Operator – oft auch „Custom Controller“ genannt – eine Anwendung, die im Kubernetes-Cluster läuft und dort Betriebsaufgaben automatisiert (siehe Abbildung 1). Dieses Operator-Pattern dient auch zum Implementieren von GitOps. Im GitOps-Operator läuft die Abstimmungsschleife (Reconciliation Loop), die den deklarativ in Git-Repositorys beschriebenen Zielzustand mit dem Istzustand des Clusters synchronisiert. Bei Unterschieden (etwa durch einen neuen Commit im Git) kümmert sich der Operator um eine Konvergenz zum Zielzustand, indem er Kubernetes-Ressourcen auf den API-Server anwendet. Erfahrungsgemäß bedarf es für einen effizienten Betrieb über die Kernfunktion hinaus weiterer Features. Dazu gehören Observability, ein Command Line Interface (CLI) oder ein User Interface (UI). Mehr dazu später im Kasten „Kriterien zur Auswahl des passenden Werkzeugs“.

Der Blog Post von Weaveworks, der 2017 den Begriff GitOps prägte, nennt auch den ersten GitOps-Operator: Flux. Mittlerweile wurde dieser als Flux v2 komplett neu geschrieben. Das dazugehörige Projekt entwickelt außer Flux und Flux v2 noch weitere Komponenten. Weaveworks hat das Projekt mittlerweile an die Cloud Native Computing Foundation (CNCF) übergeben. Dort steht es noch im untersten Reifegrad: Sandbox.

Eine Alternative zu Flux bietet ArgoCD. Es gehört zum ebenfalls bei der CNCF beheimateten Projekt Argo, das dort im zweiten Reifegrad geführt wird (Incubator), also eine Stufe fortgeschrittener als Flux. Ein umfangreicher Vergleich der beiden GitOps-Operatoren findet sich später im Artikel.

Ein neuerer Mitbewerber ist das von Rancher entwickelte Fleet. Dessen Besonderheit ist das explizite Ziel, nicht nur einen, sondern eine Flotte von Clustern verwalten zu können. Ähnlich jung und mit noch breiterem Fokus ist PipeCD. Es verspricht wie Fleet das Verwalten mehrerer Kubernetes-Cluster und bietet zusätzlich ein UI. Darüber hinaus kann es mit Terraform und einigen Services der großen Cloud-Anbieter umgehen.



Bei GitOps fragt der im Cluster laufende Operator den Sollzustand aus dem Git-Repository ab und gleicht den Istzustand damit ab. Bei CI-Ops initiiert der CI-Server die vom Entwickler eingecheckten Änderungen im Cluster (Abb. 1).

GitOps kombiniert mit CI

Ebenfalls einen breiteren Fokus – aber mit anderem Schwerpunkt – bietet Jenkins X. Anders als der Name vermuten lässt, unterscheidet es sich stark vom bekannten Jenkins-Server. Es ist kein monolithisches Werkzeug, sondern besteht aus verschiedenen Komponenten wie Tekton zum Ausführen von Pipelines und Kaniko zum Bauen von Images. Das Herz von Jenkins X ist ein CLI, das die Entwickler für die aktuelle Version 3 zusammen mit einigen grundlegenden Architekturänderungen neu geschrieben haben.

Insgesamt ist Jenkins X mächtiger als ArgoCD und Flux und dadurch schwerer in bestehende Prozesse zu integrieren. Dafür bietet es einen deutlich größeren Funktionsumfang. Es nimmt dem Benutzer viele Entscheidungen ab, verringert dadurch aber auch die Flexibilität. Jenkins X ist ein vollständiges Continuous-Integration- und Continuous-Delivery-Paket (CI/CD). Im Gegensatz dazu benötigen die reinen GitOps-Operatoren für viele Anwendungsfälle einen zusätzlichen CI-Server, der etwa Tests automatisiert sowie Images baut und in der Registry bereitstellt.

Irgendwo zwischen einem reinen GitOps-Operator und einem vollständigen CI/CD-Ansatz positioniert sich werf (gestartet als dapp, Anfang 2019 in werf umbenannt). Wie ein Operator kann es Kubernetes-Ressourcen aus Git auf einen Cluster anwenden. Allerdings läuft es außerhalb der Cluster. Damit erfüllt es nicht das häufig mit GitOps in Verbindung gebrachte Pull-Prinzip, bei dem der Cluster selbst seinen Sollzustand aus dem Git zieht. Dafür kann werf, anders als ArgoCD und Flux, auch Images bauen. Ein in Kubernetes laufender Operator ist (Stand Version v1.2 beta) in Planung.

GitOps liebt Operatoren

Ein zentraler Punkt von GitOps ist die vollständige deklarative Beschreibung des Zustandes in Git. Beim Umstellen des imperativen, klassischen CI/CD – mittlerweile

iX-TRACT

- Als logische Weiterentwicklung von Continuous-Delivery-Ansätzen erschließt GitOps eine neue Stufe der IT-Automatisierung.
- Im Kubernetes-Umfeld existieren inzwischen viele GitOps-Werkzeuge. Abseits davon bieten sich aber nur wenige Auswahlmöglichkeiten.
- Das schlanke Flux und das mit größerem Funktionsumfang ausgestattete ArgoCD sind die derzeit bekanntesten Vertreter der sogenannten GitOps-Operatoren.
- Beim Umsetzen des GitOps-Vorgehens steigt der Bedarf an weiteren Operatoren schnell.

GitOps-Werkzeuge für Kubernetes

	Flux	Flux v2	ArgoCD	Fleet	PipeCD	Jenkins X 2	Jenkins X 3	werf
Betrieb im Cluster	✓	✓	✓	✓	✓	✓	✓	–
Image Builds	–	–	–	–	–	✓	✓	✓
Pipelines	–	–	–	–	–	✓	✓	–
Multi-Cluster	–	–	–	✓	✓	–	–	–
Terraform-Support	–	–	–	–	✓	–	–	–
Projektmetriken								
Datum 1. Commit	8/2016	4/2020	2/2018	3/2020	3/2020	1/2018	7/2020	1/2016
GitHub Stars	6000	900	4800	700	320	3800	60	2200
Anzahl Contributors	250	40	370	20	20	190	10	30
Commits insgesamt	5000	1100	2900	350	1500	10000	2000	8300
Commits letzter Monat	11	116	179	16	149	0	447	237
Anzahl Issues	1500	250	2700	150	500	3800	30	600
Anzahl Pull Requests	1800	500	2700	100	1100	3700	1000	2600

zur Abgrenzung zu GitOps mancherorts als CI-Ops bezeichnet – kommen einige Fragen auf. Wie lassen sich etwa Templating-Werkzeuge wie Helm oder Kustomize weiterverwenden, die bisher der CI-Server ausgeführt hat?

Die Antwort darauf lautet meist: durch weitere Operatoren. Diese erweitern den Kubernetes-API-Server um sogenannte Custom Resource Definitions (CRDs) und lauschen dann auf Änderungen an den zugehörigen Custom Resources (CRs). In diesen CRs ist der gewünschte Zustand deklarativ beschreibbar, was perfekt zu GitOps passt.

Flux bringt etwa Helm- und Kustomize-Operatoren mit, die eine deklarative Beschreibung von Helm Releases und Kustomizations via CR erlauben. Wird eine solche CR – typischerweise vom GitOps-Operator – auf den Cluster angewendet, übernimmt der Helm- oder Kustomize-Operator das Templating oder Overlay. Alternativ zum Operator kann man das Ergebnis des Templating (beispielsweise durch eine CI-Pipeline) ins Git schreiben. Vorteile sind der geringere Aufwand für die Infrastruktur und mehr Transparenz im Git-Repository. Dem stehen umfassende und schwerer wartbare YAML-Beschreibungen im Git entgegen.

Durch das Speichern des gesamten Zustands in Git landen unweigerlich auch Secrets dort. Unverschlüsselt bieten diese dort jedoch eine große Angriffsfläche. Aber auch auf die Frage, wie sich die Ver- und Entschlüsselung mit GitOps kombinieren lässt, lautet die Antwort: durch weitere Operatoren.

Komponenten für bessere Sicherheit

Eine Option, die einfach mit GitOps zusammen funktioniert, ist der Operator Sealed Secrets von Bitnami. Der verwaltet das Schlüsselmaterial im Cluster selbst. Für die Verschlüsselung existiert ein CLI, das eine Verbindung zum Cluster benötigt.

Deutlich mehr Optionen – zulasten einer aufwendigeren Konfiguration – bietet das von Mozilla entwickelte SOPS. Hier kann das Schlüsselmaterial aus den Key-Management-Systemen (KMS) der großen Cloud-Anbieter, einem eigenen HashiCorp Vault oder aus selbst verwalteten PGP-Schlüsseln kommen. SOPS selbst enthält zwar keinen Operator, es gibt aber verschiedene Varianten, es mit GitOps zu verwenden. Flux v2 bietet native Unterstützung. Außerdem gibt es das Plug-in helm secrets, das sich mit manueller Konfiguration auch in ArgoCD verwenden lässt. Auch steht ein Operator sops-secrets von einem Drittanbieter bereit.

Einen Kompromiss zwischen Sealed Secrets und SOPS kann Kamus darstellen. Es entstand speziell für den Anwendungsfall GitOps und liefert einen Operator mit. Der kann das Schlüsselmaterial entweder selbst verwalten oder von den KMS der Cloud-Anbieter beziehen. Eine weitere Besonderheit ist, dass Kamus Secrets direkt für eine Applikation verschlüsselt. Sie werden dann von der Anwendung selbst oder einem Init-Container entschlüsselt. Dadurch liegt das Secret nie unverschlüsselt im API-Server und im Idealfall auch nicht in einer Umgebungsvariable im Container vor.

Ergänzende Werkzeuge für GitOps mit Kubernetes

	SealedSecret	SOPS	SOPS Secrets Operator	helm-secrets	Kamus	Kubernetes External Secrets	External Secret Operator	Vault Secrets Operator	Flagger	Argo Rollouts
Betrieb im Cluster	✓	–	✓	–	✓	✓	✓	✓	✓	✓
selbst verwaltete Schlüssel	✓	✓	siehe SOPS	siehe SOPS	✓	–	–	–	–	–
Secrets aus HashiCorp Vault	–	✓	siehe SOPS	siehe SOPS	–	✓	–	✓	–	–
Secrets aus proprietären KMS	–	✓	siehe SOPS	siehe SOPS	✓	✓	✓	–	–	–
Projektmetriken										
Datum 1. Commit	6/2017	8/2015	5/2019	4/2017	6/2018	12/2018	8/2018	8/2019	9/2018	11/2018
GitHub Stars	2900	6800	70	100+1200*	700	1500	170	280	2700	720
Anzahl Contributors	60	80	<10	30	140	80	20	10	80	70
Commits insgesamt	700	100	90	350	1000	340	520	210	2030	600
Commits letzter Monat	12	0	0	16	2	10	11	11	25	29
Anzahl Issues	250	460	20	10+100*	130	250	90	30	370	360
Anzahl Pull Requests	270	360	30	70+70*	480	350	70	50	440	610

* Fork, Vorgänger eingestellt

Infrastrukturnahe Werkzeuge

	Cluster API	PipeCD	Terraform Cloud Operator for Kubernetes	terraform-controller	Atlantis	AWX	Weave Ignite
Anwendungsfall	K8s-Cluster-Set-up	Terraform und weitere	Terraform	Terraform	Terraform	Ansible	VM-Management
Betrieb im Cluster	✓	✓	✓	✓	optional	–	–
Projektmetriken							
Datum 1. Commit	3/2016	3/2020	11/2019	11/2018	5/2017	2/2013	5/2019
GitHub Stars	1500	320	170	250	3200	9000	1500
Anzahl Contributors	290	20	10	10	140	50	40
Commits insgesamt	4900	1500	90	100	1700	28000	1600
Commits letzter Monat	53	149	8	0	23	325	30
Anzahl Issues	1600	500	40	30	700	5300	350
Anzahl Pull Requests	2500	1100	40	45	660	3800	420

Verwendet man ohnehin ein externes KMS, gibt es noch weitere Optionen, wie den ursprünglich von GoDaddy gestarteten Operator kubernetes-external-secrets und den externalsecret-operator von Container Solutions. Wer HashiCorp Vault einsetzt, hat zudem die Option auf den Operator Vault Secrets. Der funktioniert ähnlich dem Sealed Secrets Operator, ruft aber statt der Verwaltung von eigenem Schlüsselmaterial die Secrets aus Vault ab. Einen Überblick über die Verbreitung von Tools zum Secrets Management bietet der Technology Radar der CNCF vom Januar 2021 (siehe ix.de/zc9k).

Weitere Anwendungsfälle für ergänzende GitOps-Operatoren sind die Realisierung von Deployment-Strategien wie Canary Releases, A/B-Tests und Blue/Green Deployments – mittlerweile zusammengefasst unter dem Begriff „Progressive Delivery“. Hierfür reichen die Mittel der meisten GitOps-Operatoren nicht aus. Als Lösung bietet sich flagger an. Das von Weaveworks initiierte Tool wird mittlerweile als Teil des Flux-Projekts weiterentwickelt. Auch im Argo-Projekt findet sich ein Operator für diesen Anwendungsfall: Argo Rollouts. Beide bieten CRs zum Realisieren der Progressive-Delivery-Strategien im Zusammenspiel mit diversen Ingress-Controllern und Service Meshes.

Infrastrukturnahe Werkzeuge

Der Begriff GitOps kam im Kontext von Anwendungs-Deployments in Kubernetes auf. Für diesen Fall sind diese Werkzeuge gut einsetzbar. Sie sind allerdings nicht darauf beschränkt. So lassen sich GitOps-Operatoren auch für das Ausrollen von Kubernetes-Clustern verwenden. Ein Szenario ist die Nutzung der Kubernetes Cluster API – gestartet als kube-deploy, 2018 in Cluster API (CAPI) umbenannt. Dies lässt sich wie folgt umsetzen: In einem Managementcluster läuft ein GitOps-Operator, der die durch die CAPI definierten CRs aus Git auf den Cluster anwendet. Ein ebenfalls im Cluster laufender Infrastructure-Provider liest diese CRs und wendet sie auf einen Zielcluster an.

Sobald dieser Cluster steht, stellt sich die Frage, wie man dort Anwendungen ausrollt. Flux und ArgoCD können dazu direkt auf den API-Server zugreifen. Bei anderen Operatoren wie Fleet und PipeCD ist die Architektur speziell für diesen Fall entworfen. Sie bringen je eine Komponente für den Management- und den Zielcluster mit: Bei Fleet verbindet sich ein Manager-Operator mit den Agent-Operatoren, bei PipeCD eine Control Plane mit Daemons. Dies erfordert also keinen Zugriff auf den API-Server von außen.

Über das Erstellen von Kubernetes-Clustern hinaus entstehen auch zunehmend Möglichkeiten, verschiedene Infrastructure-as-Code-Werkzeuge (IaC) wie Terraform mit GitOps zu nutzen. Wie erwähnt bietet PipeCD Unterstützung für Terraform. Von offizieller Seite bietet dessen Hersteller HashiCorp mittlerweile auch einen Terraform Kubernetes Operator an. Der benötigt allerdings Zugang zur HashiCorps Terraform Cloud. Alternativ gibt es Operatoren von Drittanbietern, die ohne Terraform Cloud auskommen, etwa der von Rancher. Allerdings befindet sich der im Alphastadium.

Eine weiter verbreitete Alternative für GitOps mit Terraform ist Atlantis: Beim Erzeugen eines Pull Requests erstellt es auf Basis der darin enthaltenen Terraform-Dateien im Git-Repository den Terraform-Plan und fügt diesen dem Pull Request als Kommentar hinzu. Nach dem Merge wendet es den Terraform-Plan an. Atlantis ist kompatibel mit verschiedenen Git-Anbietern. Es lässt sich flexibel hosten: als Binary, als Docker-Image und als Helm Chart für Kubernetes. Damit ist es eines der wenigen Werkzeuge, das auch für GitOps ohne Kubernetes interessant ist. Ebenfalls unabhängig von Kubernetes ist Ansible Tower (und damit dessen Open-Source-Upstream AWX). Red Hat sieht den Funktionsumfang als vergleichbar mit einem GitOps-Operator an.

Das Beispiel Atlantis steht gewissermaßen sinnbildlich für GitOps außerhalb von Kubernetes. Ohne eine Plattform wie Kubernetes, in der sich ein Operator betreiben lässt, ist das Pull-Prinzip von GitOps schwieriger umsetzbar. Was bleibt, ist oft das Thema „Operations by Pull Request“. Das Erstellen, Ändern oder Mergen von Pull Requests stößt allerdings auch bei CI-Ops eine CI/CD-Pipeline an. Da stellt sich die Frage, ob also der Einsatz von Pull Requests für den Betrieb ausreicht, um als GitOps bezeichnet zu werden. Die Abgrenzung wird hier schwieriger. Tatsächlich bezeichnet Atlantis sich selbst nicht als GitOps-Werkzeug. Es gibt aber andere Beispiele wie die Terraform-Alternative Pulumi. Es arbeitet zusammen mit CI-Werkzeugen wie GitHub Actions und GitLab CI und kann auf den zugehörigen Plattformen Pull Requests kommentieren. Pulumi bezeichnet dies als GitOps.

Abschließend darf bei einer Auflistung der infrastrukturnahen GitOps-Werkzeuge eines nicht fehlen: das ebenfalls von Weaveworks gestartete Ignite. Es bietet die Möglichkeit, virtuelle Maschinen (VMs) per GitOps zu verwalten. Dazu läuft auf dem physischen Host ein Daemon, der VMs entsprechend einer Beschreibung in einem Git-Repository starten und stoppen kann. Als Virtualisierer dient das ursprünglich von AWS gestartete Firecracker.

So viel und doch zu wenig

Ironischerweise gibt es trotz der Fülle an Werkzeugen Anwendungsfälle, die noch nicht mit GitOps automatisierbar sind. Bei infrastrukturnahen GitOps-Werkzeugen ist die Auswahl deutlich kleiner als bei denen für Kubernetes. Je weiter man sich der physischen Infrastruktur annähert, desto

geringer wird die Unterstützung durch entsprechende Tools. Beispiele aus dem Betrieb, die sich so noch nicht per GitOps erledigen lassen, sind das Starten physischer Maschinen, etwa via Preboot Execution Environment (PXE), oder das Firmware-Upgrade in Geräten wie Switches.

Doch auch bei den ergänzenden Operatoren lassen sich noch nicht alle Anwendungsfälle im Betrieb vollständig mit GitOps automatisieren. Ein Beispiel ist der Horizontal Pod Autoscaler in Kubernetes. Um mit GitOps kompatibel zu sein, müsste dieser die Änderung der Anzahl von Replicas in ein Git-Repository schreiben, statt sie direkt an den API-Server zu senden.

Ein weiteres, noch aufwendigeres Beispiel ist der Themenkomplex Persistenz, Backup, Restore und Disaster Recovery. Das Erstellen von Backups ist zwar mit Operatoren wie Velero zuverlässig automatisierbar. Das Wiederherstellen erfordert allerdings einen manuellen Eingriff. Auf den im GitOps-Repository beschriebenen Zustand kann zwar zurückgerollt werden, aber das stellt den durch den Backup-Operator gespeicherten Zustand nicht wieder her. Umgekehrt setzt das manuelle Wiederherstellen mittels Backup-Operator den Zustand im GitOps-Repository nicht zurück.

Trotz der Tool-Fülle gibt es also noch Lücken im Funktionsumfang der GitOps-Werkzeugkette. Bei der derzeitigen Entwicklungsdynamik ist hier Zuversicht angebracht, dass sich diese in absehbarer Zeit schließen werden. Generell basieren viele der verfügbaren GitOps-Werkzeuge auf Kubernetes. Selbst für das Aufsetzen der Cluster oder anderer Infrastruktur wird ein Kubernetes-Cluster benötigt. Wer GitOps ohne Kubernetes umsetzen will, betritt wenig ausgetretene Pfade.

Aus dieser Tool-Menge gilt es nun das richtige für die eigenen Anforderungen zu finden. Die Entscheidung hängt dabei stark vom Anwendungsfall ab, der Kasten „Kriterien zur Auswahl des passenden Werkzeugs“ zeigt, worauf man achten sollte.

Operatoren im Vergleich: ArgoCD versus Flux v2

Ein Beispiel kann die im Kasten aufgezählten Kriterien wesentlich besser veranschaulichen. Da liegt es nahe, die beiden bekanntesten GitOps-Operatoren ArgoCD und Flux v2 miteinander zu vergleichen. Beide Projekte unterliegen einer dynamischen Weiterentwicklung, sodass der hier vorliegende Vergleich nur eine Momentaufnahme sein kann.

Für einen ersten Überblick über Gemeinsamkeiten und Unterschiede lohnt sich ein Blick auf die Featurelisten. Folgende Fähigkeiten bringen sowohl Flux v2 als auch ArgoCD mit:

- RBAC;
- Multi-Tenancy;
- Observability (Health Status, Notifications, Metriken);
- CLI für Automatisierung und CI-Integration;
- Start der Synchronisation per Webhook;
- Aufruf von Webhooks bei Events (ArgoCD: Hooks, Flux: Notifcations);
- Rollback/Roll-anywhere zu bestimmten Commits im Git-Repository;
- Unterstützung für Helm und Kustomize;
- Multi-Cluster-Unterstützung;
- Ausführung der Container als unprivilegierter User;
- Security Context konfigurierbar.

Funktionen nur in ArgoCD:

- Unterstützung für Ksonnet und Jsonnet und weitere per Plug-in-Mechanismus;
- Weboberfläche zur Administration und Überwachung von Applikationen in Echtzeit;
- SSO-Integrationen für UI und CLI.

Funktionen nur in Flux v2:

- Definition von Abhängigkeiten in Helm- und Kustomize-Applikationen;
- Unterstützung von SOPS;
- automatisches Update der Image-Version im Git-Repository;
- Authentifizierung des CLI per Kubeconfig.

Installation und Konfiguration

ArgoCD lässt sich entweder über einfache Kubernetes-Ressourcen installieren und anschließend patchen oder über ein konfigurierbares Helm Chart ausrollen. Alle gängigen Einstellungen sind vorhanden, etwa Ingress und Service-Accounts. Es können zudem Repositorys, Projekte, Applikationen und verschiedene SSO-Integrationen für das Deployment vorkonfiguriert werden. Auch die RBAC-Konfiguration ist umfangreich. Richtlinien, Rollen und Gruppen lassen sich selbst definieren.

Eine weitere Variante der Installation bietet der zusätzliche ArgoCD-Operator. Dieser erlaubt die Konfiguration und Installation der eigentlichen ArgoCD-Komponenten per CRD. Nicht dokumentiert ist, wie man ArgoCD selbst per GitOps konfiguriert. Dies ist zwar denkbar, beispielsweise mittels des ArgoCD-Operators. Ob dies zuverlässig funktioniert und vor allem auch im Fehlerfall einen weiteren Betrieb per GitOps ermöglicht, bleibt herauszufinden.

Das Ausrollen der Flux-v2-Operatoren erfolgt primär über das CLI. Darüber lassen sich aber auch alle erforderlichen Kubernetes-Ressourcen generieren und anschließend auf den Cluster ausbringen. Es gibt derzeit kein Helm Chart, was die gängigen Ausbringungsmechanismen einschränkt. Bei der ersten Installation richtet Flux ein Git-Repository ein. Ab diesem Punkt kann man auch den Operator mit GitOps konfigurieren.

Das CLI ist umfangreich und erlaubt die Steuerung aller Aspekte des Flux-v2-Portfolios. Man kann damit alle Flux-v2-spezifischen Ressourcen erzeugen, löschen, lesen, starten und stoppen. Damit eignet sich Flux v2 besonders gut für Scripting. Das mag zwar zunächst widersprüchlich zu GitOps erscheinen, aber in der Praxis findet häufig ein schrittweiser Umstieg statt. Hierbei kann das Erzeugen von Res-

Darf es etwas mehr sein?

Es existieren noch viele weitere Werkzeuge, die sich entweder selbst mit GitOps in Verbindung bringen oder oft in diesem Kontext genannt werden. Die Autoren stehen gern für Diskussionen über das Thema im GitOps-Themenforum bei heise online bereit (siehe ix.de/zc9k). Um der häufig nach einer Marktübersicht gestellten Frage „Warum wird Programm ‚X‘ nicht erwähnt?“ etwas entgegenzutreten, hier noch einige Gründe, warum dieser Artikel bestimmte Kandidaten nicht berücksichtigt. Außen vor bleiben Werkzeuge, die

- ausschließlich imperativ oder im CI/CD-Prozess aufgerufen werden, also keine Reconciliation Loop haben (beispielsweise Templating-Werkzeuge), auch wenn sie sich zum Teil mit dem Attribut GitOps schmücken;
- nicht mehr aktiv weiterentwickelt werden;
- generell nicht für den Produktiveinsatz gedacht sind;
- einen stark beschränkten Anwendungsfall haben, wie speziell auf einen Cloud-Anbieter zugeschnitten zu sein;
- proprietär sind;
- noch sehr neu und deshalb noch nicht sehr weit verbreitet sind.

Details zur im ersten Punkt genannten Standardisierung des Begriffs GitOps liefert der Artikel „Finger weg“ ab Seite 42 [1].

Kriterien zur Auswahl des passenden Werkzeugs

Im Kubernetes-Umfeld existieren derzeit die meisten GitOps-Werkzeuge. Kein Wunder, schließlich ist das auch der älteste und ausgereifteste GitOps-Anwendungsfall. Dieser Kasten fasst – auch auf Basis eigener Erfahrungen der Autoren – wichtige Anforderungen für Entscheidungskriterien zusammen. Viele dieser Kriterien gelten jedoch nicht ausschließlich für den Einsatz mit Kubernetes. Insofern können sie auch für den Einsatz von GitOps in anderen Umfeldern hilfreich sein.

Installation und Konfiguration des GitOps-Operators

Bekanntlich existieren viele Möglichkeiten, Anwendungen auf einen Kubernetes-Cluster auszurollen. Das gilt auch für GitOps-Operatoren. Gängige Optionen sind das Ausrollen per Helm Chart, CLI oder als einfache Kubernetes-Ressourcen. Ein weiterer Aspekt ist das Einrichten des Operators im Betrieb. Ein interessanter Punkt dabei: Lässt sich der Operator selbst auch per GitOps konfigurieren?

Zusätzlich stellt sich die Frage, welche Teile sich zur Laufzeit anpassen lassen (beispielsweise durch CRs – Custom Resources) und welche Teile einen Neustart des Operators erfordern. Erlaubt das Werkzeug eine Multi-Tenancy-Lösung? Für deren Umsetzung kann ein Multi-Cluster-fähiger Operator hilfreich sein. Operatoren können diese durch eigene Komponenten im Zielcluster unterstützen oder direkt mit dem API-Server kommunizieren. Je nach Infrastruktur kann der direkte Zugriff darauf erwünscht sein oder nicht.

Konfiguration der Applikationen

Damit der GitOps-Operator den Cluster mit einem Git-Repository synchronisieren kann, muss dieses definiert und konfiguriert sein. Dazu gilt es, vorhandene Projektstrukturen mit dem eingesetzten Operator abzubilden. Weiter ist interessant, ob er mehrere Git-Repositorys einbinden kann und ob sich diese im Betrieb hinzufügen und konfigurieren lassen. Neben einfachen Kubernetes-Ressourcen existieren noch weitere Wege, Applikationen in einem Kubernetes-Cluster auszurollen. Dazu gehören Kustomize, Helm, Ksonnet oder Jsonnet.

Welche Applikationstypen ein Operator unterstützt und ob sie den vorhandenen Bedarfen entsprechend konfiguriert werden können, kann die Entscheidung für dessen Einsatz beeinflussen. Eine weitere Rolle können Hooks spielen, also die Möglichkeit, auf Events während des Deployments zu reagieren, um Nachrichten zu senden, Überprüfungen vorzunehmen oder das Deployment zu beeinflussen. Auch die Abbildung von Abhängigkeiten zwischen Ressourcen kann relevant sein. Beispiele hierfür sind die Anwendung von CRDs vor zugehörigen CRs oder dass eine Datenbank läuft, bevor die Anwendung gestartet wird.

Vereinzelt bieten Werkzeuge auch weiter gehende Möglichkeiten zur Automation, beispielsweise beim Bauen von Images, beim Schreiben neuer Image-Versionen ins Git oder sogar beim Ausführen ganzer Pipelines.

UI

Eine grafische Oberfläche erlaubt den bequemen Zugriff auf die Konfiguration des Operators und die Applikationsressourcen. Sie ermöglicht eine Fehleranalyse und -behandlung ohne Zugang zum Cluster, indem Clusterobjekte überwacht und Manifeste manipuliert werden können. Hier ist jedoch zu beachten, dass die Änderungen nicht ins Git-Repository synchronisiert, sondern nur die entsprechenden CRs im Cluster verändert werden. Ein Single Sign-on (SSO) ist von Vorteil, da sich so ein vorhandenes User-Management einsetzen lässt, Benutzern den Zugang zum UI zu ermöglichen.

CLI

Mit einem CLI lassen sich Skripte entwickeln, die GitOps-Prozesse automatisieren oder in CI-Pipelines integrieren. Denkbar ist das Ausrollen neuer Cluster mitsamt Operatoren oder das Antriggern der Reconciliation Loop in bestehenden Workflows. Je nachdem, wie weitgehend das CLI eingesetzt werden soll, gewinnt dessen Funktionsumfang an Bedeutung.

Autorisierung

Für GitOps ist die Vergabe von Berechtigungen von zentraler Bedeutung, da prinzipiell alles, was im Git-Repository landet, auch auf den Cluster angewendet wird. So könnten kritische oder sicherheitsrelevante Objekte verändert werden und den Cluster kompromittieren oder in einen unbenutzbaren Zustand versetzen. Eine Gegenmaßnahme: den Zugriff des Operators auf bestimmte Ressourcentypen beschränken. Hierfür eignet sich die Role-based Access Control (RBAC), um den Operator nicht mit den Rechten eines Cluster-Admins auszurollen. Zusätzliche Einschränkungen durch Allow- und Deny-Listen auf Ressourcen und Ressourcentypen können die Sicherheit weiter verbessern.

Observability

Wegen der Asynchronität von GitOps spielen Feedback-Mechanismen eine bedeutende Rolle. Denn erst nachdem der Operator die Ressourcen auf den Cluster ausgebracht hat, kann man wissen, ob das Deployment erfolgreich war. Zum Erhalt dieser Benachrichtigungen benötigt man zusätzliche Werkzeuge (beispielsweise Chat, E-Mail, Metriken oder Commit Status). Hier ist zu betrachten, welche Werkzeuge zum Einsatz kommen können und welchen Aufwand es erfordert, sie in den eigenen Workflow zu integrieren.

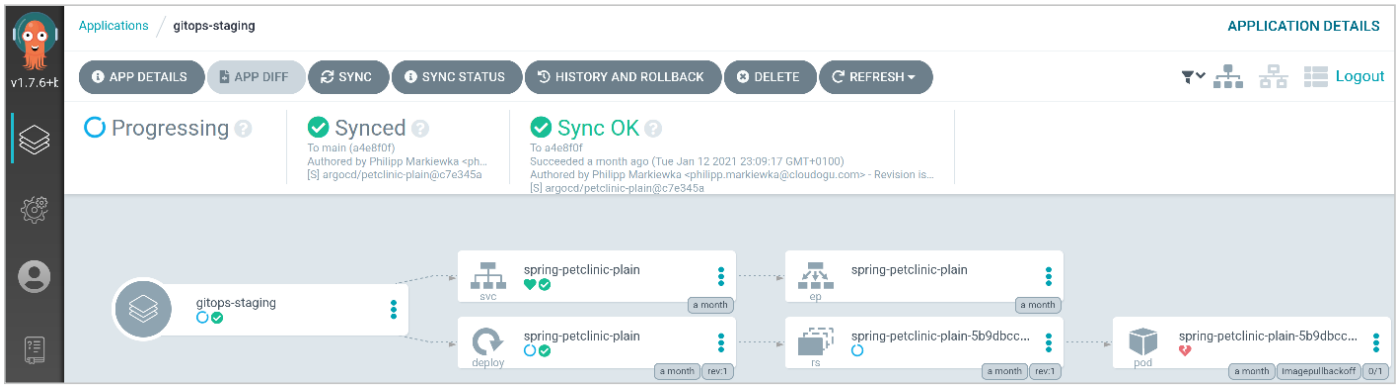
sourcen, die dann in ein Git-Repository eingecheckt werden, über das CLI erfolgen. So lässt sich GitOps in bestehende CI/CD-Prozesse integrieren.

Konfiguration der Applikationen

ArgoCD bildet den Sollzustand intern durch Projekte und Applikationen ab. In Projekten lassen sich Repositorys, Cluster und Berechtigungen festlegen. Einem Projekt kann man wiederum beliebig viele Applikationen zuweisen, die Zugriff auf die Cluster und Repositorys des Projekts erhalten. Diese Applikationen und Projekte speichert ArgoCD als CRs im Cluster. In einer Applikation finden sich dann das Repository und der Pfad zu einem Deployment. Zudem kann man dort Konfigurationen für Helm, Kustomize et cetera in deklarativer Form beschreiben. Dieses über die Applikation definierte Deployment wendet ArgoCD dann auf den Cluster an. ArgoCD speichert Helm-Releases allerdings nicht als solche im Cluster, sondern wandelt sie mittels `helm template` in einfache Kubernetes-Ressourcen um und wendet diese auf den Cluster an. Im Gegensatz zu Flux lassen sich die Helm-Releases also nicht per `helm ls` aufrufen und es gibt keine Historie der Releases im Cluster.

Die Definition aller relevanten Elemente erfolgt über CRDs, womit diese selbst per GitOps konfigurierbar sind. Konkret bedeutet das, dass man die als CR definierten Projekte, Applikationen et cetera in einem eigenen Git-Repository pflegen kann.

Bei Flux v2 bilden die Repositorys die zentrale Stelle. Sie dienen sowohl dem Deployment von Workloads als auch der Konfiguration von Flux selbst. Als Workloads kann Flux Kubernetes-Ressourcen, Helm Release CRs und Kustomize CRs ausrollen. Die Helm Release CRD und Kustomize CRD bieten zudem alle bekannten Funktionen dieser Werkzeuge in deskriptiver Form an. Da auch die Git-Repositorys



Benutzer von ArgoCD können den Status von Applikationen und deren Ressourcen in der grafischen Benutzeroberfläche einsehen (Abb. 2).

als CR definiert sind, lassen sie sich ebenfalls über GitOps verwalten.

UI

ArgoCD bringt eine Weboberfläche für Administration und Monitoring der Applikationen mit. Da GitOps ein asynchrones Verfahren ist, erhält man bei Änderungen an Deployments kein sofortiges Feedback. Abhilfe schaffen hier die mitgelieferten Notification-Mechanismen (siehe Abschnitt Observability). Sie geben Rückmeldung über verschiedene Kanäle wie Chat oder E-Mail. Zusätzlich bietet ArgoCD über das UI die Möglichkeit, die Deployments auch in Echtzeit zu beobachten (siehe Abbildung 2). Wenn Teile des Deployments fehlschlagen, kann man diese über das UI identifizieren und Fehlermeldungen und Logs einsehen.

Damit kann ein Entwickler – ganz ohne Zugriff auf den Cluster – seine Deployments analysieren und Fehler korrigieren. Für die Authentifizierung gibt es Schnittstellen zu gängigen Protokollen wie LDAP und OIDC. Benutzer können mit konfigurierbaren Rollen und Gruppen Zugriff auf die Projekte und Applikationen bekommen, für die sie auch zuständig sind. Für Flux v2 arbeiten die Entwickler gerade an einer Weboberfläche. Die befindet sich aber noch in einem experimentellen Zustand.

CLI

Mit dem CLI von ArgoCD lassen sich alle relevanten Funktionen bedienen. Nutzer können sowohl Objekte wie Applikationen oder Projekte erzeugen und löschen als auch Zustände verändern, wie das Zurückrollen von Applikationen oder das Antriggern von Synchronisationen. Damit eignet sich das CLI für alle Arten von Automation und Integration in CI-Pipelines. Das CLI kommuniziert mit dem ArgoCD-Server, was ein Exponieren des Kubernetes-API-Servers unnötig macht.

Auch das CLI von Flux v2 deckt alle relevanten Funktionen ab. Zusätzlich bietet es die Möglichkeit, weitere Tenants auszurollen, was beim Automatisieren einer Multi-Tenancy-Umgebung Vorteile bringen kann. Das CLI kommuniziert direkt mit dem Kubernetes-API-Server, der also von außen erreichbar sein muss.

Autorisierung

ArgoCD setzt auf eine Organisation über Projekte und Applikationen. Einem Projekt lassen sich sowohl Applikationen, Git-Repositorys, Cluster und Berechtigungen (RBAC) als auch Allow- und Deny-Listen für Ressourcentypen zuweisen. Applikationen bieten dieselben Optionen zum Einschränken der Berechtigungen. Damit können Nutzer Berechtigungen sehr einfach und feingranular sowohl auf Projekt- als auch auf Applikationsebene vergeben.

Bei Flux v2 lassen sich derzeit nur die Operatoren, Helm-Releases und Kustomization CRs per RBAC einschränken. Darüber lassen sich einfache Kubernetes-Ressourcen wie ConfigMaps und Services nur durch die Berechtigungen des Operators regeln. Auf Projektebene dürfte das zum Vergeben projektspezifischer Berechtigungen mehrere Instanzen des Flux-Operators erfordern. Alternativ lässt sich der Zugriff auf die GitOps-Repositorys auch über eine CI-Pipeline realisieren, die dann nur den Push erlaubter Ressourcen in das GitOps-Repository zulässt.

Observability

Für die Benachrichtigung über den Zustand der Synchronisation zwischen Git und Cluster bringt ArgoCD die Komponente ArgoCD Notifications mit, ein mit dem ArgoCD-Deployment ausgeliefertes Benachrichtigungssystem. Ereignisse, die beim Ausrollen von Applikationen entstehen, lassen sich damit an diverse Kanäle weiterleiten. Derzeit beherrscht ArgoCD Notifications die Kanäle SMTP, Slack, Opsgenie, Grafana, Telegram und Webhooks.

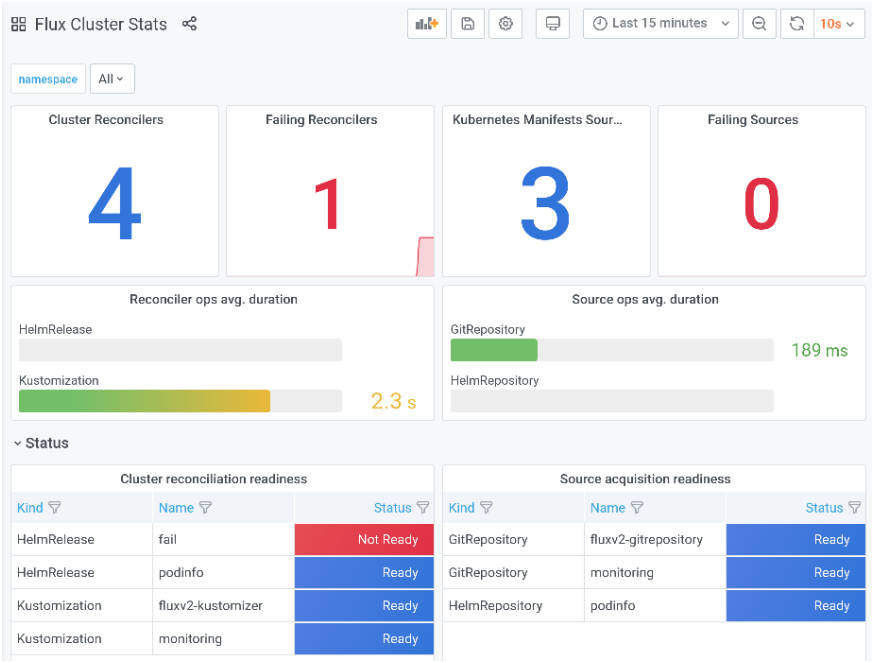
Alternativ zu ArgoCD Notifications könnte man Argo Kube Notifier und Kube Watch einsetzen. Diese sind aber zusätzlich zu ArgoCD zu betreiben. Allerdings ist ArgoCD Notifications am besten auf ArgoCD zugeschnitten, bietet nützliche Trigger und Templates und lässt sich über das projekteigene Helm Chart installieren und konfigurieren.

Zum Überwachen der Komponenten und der verwalteten Deployments exponiert ArgoCD zwei Sätze von Prometheus-Metriken: „Application Metrics“ für die Überwachung des Zustands der Synchronisierung und des Health-Zustands von Deployments sowie „API Server Metrics“ für das Überwachen der Requests und Responses an den API-Server. Darüber hinaus existieren vorgefertigte Grafana-Dashboards, die auf diesen Metriken aufbauen. Damit lässt sich sehr einfach ein Monitoring-Cockpit für das gesamte System realisieren.

Flux v2 bringt für die Observability bereits einen speziellen Controller mit, den Notification Controller. Die Konfiguration von Alerts und Notifications erfolgt dabei wie bei allen anderen Flux-Komponenten über entsprechende Custom Ressource Definitions. Über diese lassen sich Provider, also Kanäle wie Chats und Webhooks, Alert-Regeln und Empfänger einrichten. Die zugehörigen CRs können auch in einem Git-Repository gepflegt und über Flux v2 ausgebracht werden. Derzeit eignet sich der Notification Controller für die Kanäle Slack, Discord, Microsoft Teams, RocketChat und Webhooks.

Zusätzlich gibt es die Möglichkeit, den Status an einen Git Commit anzuhängen. Dies ist bei folgenden Providern möglich: GitHub, GitLab, Bitbucket und Azure DevOps. Derzeit kann Flux v2 keinen Provider per SMTP-Protokoll anbinden, um so E-Mails zu verschicken.

Zum Überwachen der Controller und Deployments steht eine Reihe von Prometheus-Metriken und Grafana-Dashboards bereit (siehe Abbildung 3). Damit lässt sich zum einen die fehlerfreie Ausführung der Controller überwachen, etwa mit Statistiken zum CPU- und Speicherverbrauch. Zum anderen kann man die Zustände aller Flux-v2-CRDs, wie Helm Releases und Kustomizations überwachen. Über diesen Weg ist es dann auch wieder möglich, Alerts per E-Mail zu verschicken. Die Definition der Alert Rules kann dabei wahlweise im Grafana oder Prometheus Alertmanager erfolgen. Prometheus und Grafana lassen sich auch über das Flux-CLI installieren und konfigurieren. In einer Multi-Tenant-Umgebung kann man darüber das Ausbringen durch Skripte automatisieren.



Auch wenn Flux noch keine Benutzeroberfläche bietet, lassen sich Metriken mit Grafana visualisieren (Abb. 3).

Fazit

Sowohl Flux v2 als auch ArgoCD bieten viele Funktionen, die auch die jahrelange Praxiserfahrung mit GitOps widerspiegeln. Beide fokussieren sich stark auf die GitOps-Kernfunktionen, was sie gut in bereits vorhandene CI/CD-Infrastruktur integrierbar macht. Wer auf der grünen Wiese anfängt, muss sich diese jedoch getrennt aufbauen.

Insgesamt bietet ArgoCD mehr Möglichkeiten in der Konfiguration (beispielsweise bei der Autorisierung) und stellt eine grafische Oberfläche bereit. Dafür ist diese allerdings zu konfigurieren, zu betreiben und bietet mehr Angriffsfläche. Hier hängt es vom Anwendungsfall ab, ob ein UI überhaupt benötigt wird und so den höheren Aufwand rechtfertigt. Für Benutzer von OpenShift relativiert sich der Aufwand, da ArgoCD als OpenShift GitOps in die Plattform integriert ist.

Flux hat dafür kleinere Features, die es in ArgoCD nicht gibt, wie die Unterstützung für SOPS und das automatische Update bei neuen Image-Versionen. Letzteres ist allerdings der Grund, warum Flux v2 noch nicht als stabile Version erschienen ist. Die Entscheidung für ein Produkt mit einer Versionsnummer 0.x bei einem so zentralen Baustein in der Lieferkette könnte schwerfallen. Hier ist allerdings bald mit einer stabilen Version zu rechnen. Eine Möglichkeit, sich selbst einen Eindruck zu verschaffen, ArgoCD und Flux v2 ohne großen Aufwand auszuprobieren und in der Praxis zu vergleichen, bietet das von den Autoren initiierte Projekt GitOps Playground (siehe ix.de/zc9k). (avr@ix.de)

Quellen

[1] Schlomo Schapiro; Finger weg; GitOps läutet die Ära des automatisierten IT-Betriebs ein; *iX* 4/2021, S. 42

[2] Weitere Hintergrundinfos zu diversen Projekten, Links zum Quellcode sowie zum Themenforum GitOps bei heise online: ix.de/zc9k

Philipp Markiewka

entwickelt und betreibt zusammen mit seinem Team per GitOps die Plattform my.cloudogu.com und gibt dabei Gelerntes gern als Autor, Trainer und Consultant weiter.

Johannes Schnatterer

macht bei Cloudogu Dev, Ops, Trainings und Consulting. Seine aktuellen Schwerpunkte sind Cloud, Container, Contiuous Delivery und GitOps.