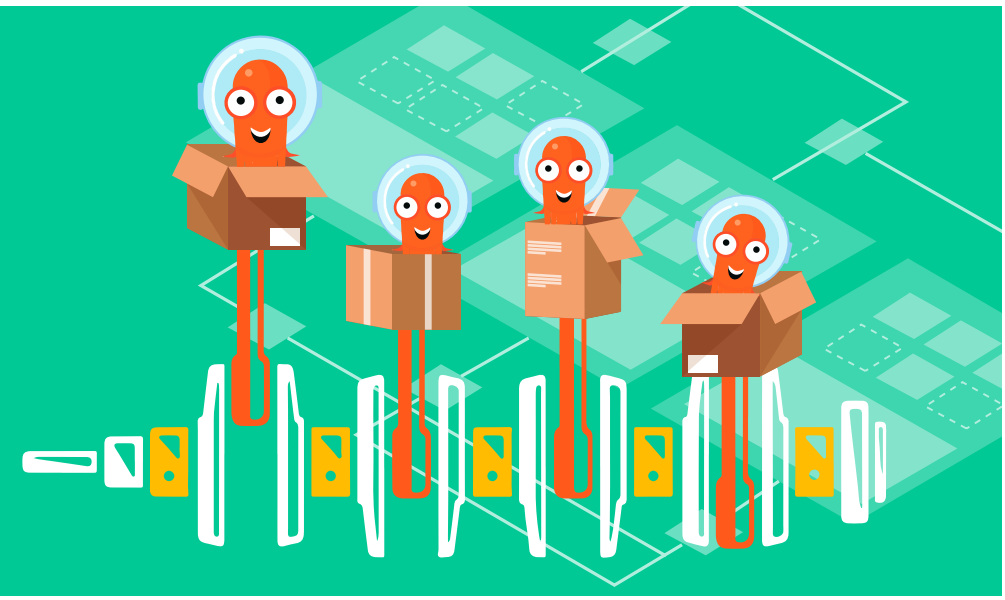


GitOps mit Crossplane, Teil 2: Basisinstallation

Für eine Infrastrukturprovisionierung mit einer durchgängigen GitOps-Pipeline sollten auch Provisionierer und GitOps-Werkzeug – sprich Crossplane und Argo CD – nach den GitOps-Regeln eingerichtet werden.

Von Jonas Hecht

Mit der Kubernetes-Erweiterung Crossplane lässt sich die Infrastruktur vollständig nach den GitOps-Prinzipien bereitstellen und dafür ein Workflow einrichten, der dem der Anwendungsentwicklung stark ähnelt, selbst bei komplexen Multi-Cloud-Szenarien. Dadurch lassen sich Anwendungs-Deployment und Infrastrukturprovisionierung mit derselben Vorgehensweise umsetzen. Was eine GitOps-Implementierung ausmacht und welche Vorteile sie bietet, hat Teil 1 des Tutorials zu GitOps mit Crossplane erläutert [1].

Der vorliegende Teil 2 stellt die Basisinstallation vor. Sie verknüpft Crossplane mit dem GitOps-Werkzeug Argo CD. Zuerst muss Argo CD fit für Crossplane gemacht und in den Managementcluster installiert werden. Danach kann Argo CD die Crossplane-Installation übernehmen (siehe Abbildung 1). Alle Codebeispiele finden sich auf GitHub (siehe ix.de/z17z).

iX-TRACT

- Nachdem der erste Teil des Tutorials die Vorteile einer durchgängigen GitOps-Pipeline sowohl für das Anwendungs-Deployment als auch für die Infrastrukturprovisionierung erläutert hat, geht es nun an die Basisinstallation.
- Die Basisinstallation verknüpft den Infrastrukturprovisionierer Crossplane mit dem GitOps-Werkzeug Argo CD.
- Vor der eigentlichen Installation auf einem mit dem Tool kind eingerichteten Managementcluster ist Argo CD auf das Zusammenspiel mit Crossplane vorzubereiten.
- Die Konfiguration von Argo CD geschieht ebenfalls deklarativ. Die Mittel dazu stellt das Kubernetes-Konfigurationswerkzeug Kustomize bereit.
- Das GitOps-Werkzeug Argo CD übernimmt dann auch die Crossplane-Installation.

Tutorialinhalt

Das Henne-Ei-Problem: der Managementcluster

Wer ein GitOps-Werkzeug wie Argo CD einsetzen will, benötigt in aller Regel einen separaten Kubernetes-Cluster, der als Schaltzentrale für die GitOps-Agenten dient und auf dem keine Anwendungen ausgerollt werden. Entsprechend heißt er Managementcluster oder Control Plane. Doch läuft man hier bereits in das klassische Henne-Ei-Problem. Denn eigentlich soll man die GitOps-Werkzeuge zum Bereitstellen der Infrastruktur nutzen, doch benötigen die Werkzeuge selbst eine Infrastruktur.

Für die erste Ausbaustufe genügt ein simpler, mit kind eingerichteter Cluster, also eine Kubernetes-in-Docker-Implementierung (alle Werkzeuge siehe ix.de/z17z). Später lässt er sich durch mächtigere Werkzeuge ersetzen. Den kind-Cluster startet man auf der Kommandozeile. Da im weiteren Verlauf noch weitere Werkzeuge zum Einsatz kommen, empfiehlt es sich, sie bereits zu diesem Zeitpunkt ebenfalls zu installieren (siehe Kasten „Verwendete Werkzeuge“). Unter macOS erledigt das der Befehl

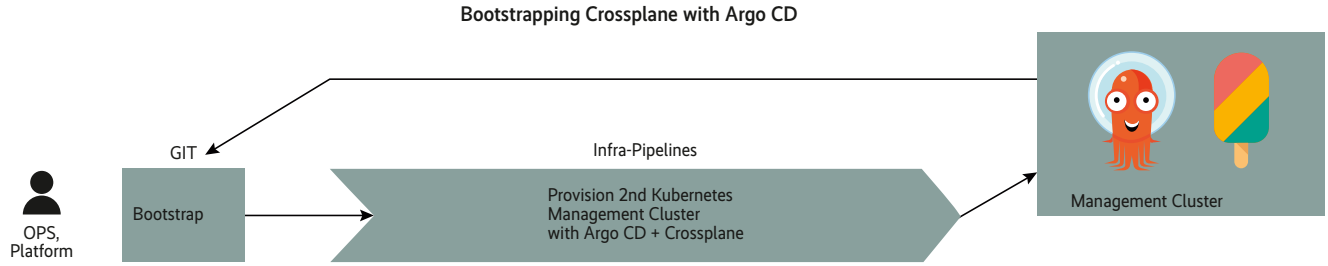
```
brew install kind helm kubectl ↵
    kustomize argocd derailed/k9s/k9s
```

Andere Paketmanager auf anderen Betriebssystemen verlangen unter Umständen etwas andere Paketnamen.

Der obige Befehl installiert die CLI-Werkzeuge für Kubernetes-Pakete Helm und für Kubernetes-Konfigurationen Kustomize, zudem `argocd` und `kubectl`. Außerdem ist es von Vorteil, das Kubernetes-Managementtool k9s griffbereit zu haben. Darüber hinaus muss für die Ausführung von `kind` eine Installation von Docker vorhanden sein. Nun steht alles bereit, um mit dem Befehl

```
kind create cluster --image ↵
      kindest/node:v1.32.1 --wait 5m
```

lokal einen Cluster zu starten.



Zur Integration von Crossplane und Argo CD sind es nur wenige Schritte (Abb. 1).

Argo CD in den Managementcluster installieren

Doch bevor man mit der Installation von Argo CD beginnen kann, sind einige Anpassungen an seiner Konfiguration notwendig. Das wirft die Frage auf, wie man Argo CD sinnvoll installieren und gleichzeitig die Installationsmanifeste so anpassen kann, dass sie flexibel und GitOps-freundlich bleiben. Idealerweise soll Renovate das Ganze hinterher auch noch automatisch aktuell halten können.

Einen Weg zeigt das Argo-CD-Team selbst auf: Es nutzt Kustomize, um seine eigenen Live-Instanzen von Argo CD kontinuierlich auszurollen. Die Konfiguration ist auf GitHub einsehbar (siehe ix.de/z17z).

Der Rückgriff auf Kustomize eröffnet einen eleganten Weg, die Argo-CD-ConfigMaps deklarativ zu ändern und gleichzeitig die anderen Installationsmanifeste unberührt zu lassen. Denn die Default-Installation von Argo CD, wie sie etwa in der install.yaml auf GitHub zu finden ist, sollte man möglichst beibehalten, um auch zukünftige Versionen von Argo CD nicht manuell installieren zu müssen (alle Manifeste siehe ix.de/z17z). Gleichzeitig erlaubt diese Methode die Nutzung von Renovate innerhalb der CI/CD-Pipeline, die das Bootstrapping von Argo CD und Crossplane später übernimmt.

Den richtigen Ort für die Anpassungen wählen

Um Kustomize für die Installation von Argo CD zu verwenden, bietet es sich an, ein Verzeichnis argocd/install zu erstellen, und zwar innerhalb eines neuen Ordners, der als Basis für das hier vorgestellte Verfahren dient. In ihm legt man eine Datei kustomization.yaml an (siehe Listing 1).

Unter dem Parameter `resources` findet sich ein Link zum Manifest für die Argo-CD-Installation, gefolgt von einem Versions-Tag. Auf diese Weise kann Renovate mit der Kustomize-Datei arbeiten und wird – einmal konfiguriert – fortan das Aktualisieren übernehmen, sollte eine neue Argo-CD-Version erscheinen. Das Beispielprojekt auf GitHub verwendet Renovate deshalb auch fleißig zusammen mit GitHub Actions.

Außerdem verweist der Abschnitt `resources` auf eine Datei argocd-namespace.yaml. Sie übernimmt das Erstellen des notwendigen Namespace `argocd`. Man legt sie ebenfalls im Ordner `argocd/`

Verwendete Werkzeuge

Die Beispielinstallation verwendet eine Vielzahl unterschiedlicher Open-Source-Werkzeuge. Hier eine Übersicht.

Das GitOps-Werkzeug **Argo CD** stellt die Agenten und Befehle für das automatische Installieren, Überwachen und Aktualisieren der ihm anvertrauten Ressourcen bereit. Die Agenten überwachen die Vorgaben in den Repositorys und spielen Änderungen automatisch auf die Systeme auf.

Crossplane ist eine Kubernetes-Erweiterung zum deklarativen Verwalten von Cloud-Infrastrukturen. Mit ihm lassen sich Kubernetes-Cluster in universelle Control Planes verwandeln, die Kubernetes-Anwendungscluster und ihre Ressourcen verwalten.

Docker isoliert Anwendungen mithilfe der Containervirtualisierung. Es stellt Anwendungen als Container-Images bereit, die sich leicht transportieren und installieren lassen und alles enthalten, was zum Ausführen einer Anwendung erforderlich ist: Code, Laufzeit, Systemtools, Systembibliotheken und Einstellungen.

Der Kubernetes-Paketmanager **Helm** dient der Verwaltung von Kubernetes-Anwendungen. Sie lassen sich mithilfe von Helm Charts definieren, installieren und aktualisieren. Helm Charts sind Sammlungen von YAML-Dateien, die die Struktur und Konfiguration einer Kubernetes-Anwendung definieren. Ein Chart kann alles enthalten, was für den Betrieb einer Anwendung notwendig ist, einschließlich Deployments, Services, ConfigMaps und Secrets. Charts lassen sich versionieren, teilen und auf öffentlichen oder privaten Servern hosten.

Das Kubernetes-Managementwerkzeug **k9s** für die Konsole überwacht bereits installierte Kubernetes-Cluster kontinuierlich auf Änderungen und bietet Befehle zur Interaktion mit den beobachteten Ressourcen. Es soll die Navigation, das Monitoring und die Verwaltung der Anwendungen vereinfachen.

Das Testwerkzeug **kind** zählt zu den Kubernetes-eigenen Tools für die Konsole. Es dient dazu, auf dem lokalen Rechner einen Kubernetes-Cluster auszuführen, dessen Knoten aus Docker-Containern bestehen. Deshalb benötigt kind eine konfigurierte und funktionierende Docker-Installation. kind wurde ursprünglich zum Testen von Kubernetes selbst entwickelt.

Der Kubernetes-Manager **kubectl** ist wohl das wichtigste und verbreitetste Kubernetes-eigene Werkzeug für die Konsole. Mit ihm lassen sich Befehle auf einem Kubernetes-Cluster ausführen, etwa Anwendungen bereitstellen, Clusterressourcen überwachen und verwalten oder Logs einsehen. Die Befehle folgen alle einer klaren Syntaxregel und lassen sich auf jede Art von Ressource anwenden. Sie alle beginnen mit `kubectl`.

Mit dem Kubernetes-Konfigurationstransformationstool **Kustomize** lassen sich YAML-Konfigurationsdateien anpassen, ohne die als Vorlagen dienenden Originaldateien zu verändern.

Renovate ist ein Werkzeug zum Aktualisieren von Software. Es automatisiert Software-Updates, indem es Pull Requests erstellt, sobald es neuere Versionen im Repository findet.

Listing 1: kustomization.yaml

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- github.com/argoproj/argo-cd//manifests/cluster-install?ref=v2.13.4

- argocd-namespace.yaml

## changes to config maps
patches:
- path: argocd-cm-patch.yaml

namespace: argocd
```

Listing 2: argocd-namespace.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: argocd
```

Listing 3: Die Exklusion der ProviderConfigUsage-CRDs konfigurieren

```
  # Set Resource Exclusion (see https://docs.crossplane.io/knowledge-base/↵
                        integrations/argo-cd-crossplane/#set-resource-exclusion)
  resource.exclusions: |
    - apiGroups:
      - "*"
      kinds:
      - ProviderConfigUsage
```

Listing 4: argocd-cm-patch.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
data:
  # Set Resource Tracking Method
  application.resourceTrackingMethod: annotation
  # Set Resource Exclusion (see https://docs.crossplane.io/knowledge-base/↵
                        integrations/argo-cd-crossplane/#set-resource-exclusion)
  resource.exclusions: |
    - apiGroups:
      - "*"
      kinds:
      - ProviderConfigUsage
```

Listing 5: Chart.yaml

```
apiVersion: v2
type: application
name: crossplane
version: 0.0.0 # unused
appVersion: 0.0.0 # unused
dependencies:
  - name: crossplane
    repository: https://charts.
crossplane.io/stable
    version: 1.18.2
```

`install` an (siehe Listing 2). Dank dieses simplen Manifestes und seiner Integration in die kustomization.yaml muss der Namespace nicht explizit angelegt werden; das übernimmt Kustomize.

Argo CD fit für Crossplane machen

Darüber hinaus verweist der Abschnitt `patches` auf eine Patchdatei namens argocd-cm-patch.yaml zum Anpassen der Argo-CD-ConfigMaps. Mit solchen Patch-Files bietet Kustomize die Möglichkeit, die eigentlichen Anpassungen an den Argo-CD-Installationsmanifesten deklarativ vorzunehmen. Deshalb sollte man zusätzlich eine solche Patchdatei argocd-cm-patch.yaml im Ordner argocd/install anlegen.

Eine Aufstellung, welche Anpassungen an die Argo-CD-Konfiguration notwendig sind, findet sich in der Crossplane-Dokumentation. Doch nicht jeder der dort beschriebenen Punkte muss sofort angegangen werden. Beispielsweise kann man die Konfiguration des Health-Status ignorieren, da die meisten Crossplane-Provider ihn bereits mitbringen.

Deshalb genügt es, sich vorerst auf zwei andere Punkte zu konzentrieren: die Konfiguration des annotationsbasierten Ressourcentrackings in Argo CD und die Exklusion der Crossplane-generierten `ProviderConfigUsage`-CRD (Custom Resource Definition).

Annotationsbasiertes Ressourcentracking in Argo CD

Argo CD kennt mehrere Wege, Ressourcen zu tracken. Mit dem Tracking hält sich Argo CD über freigegebene Änderungen in der Single Source of Truth auf dem Laufenden, um sie auf die vorgesehenen Zielsysteme zu ziehen. Für die Integration von Crossplane benötigt Argo CD einen speziellen Trackingmodus.

Per Default steuert das Label `app.kubernetes.io/instance` das Ressourcentracking in Argo CD. Seit der Version 2.2 bietet Argo CD weitere Möglichkeiten an. Eine davon ist das annotationsbasierte Tracking.

Diese Variante hat einige Vorteile. Sie definiert, wem eine Ressource zuzuordnen ist. Damit verhindert sie auch, dass sich andere Kubernetes-Tools wie Crossplane mit Argo CD ins Gehege kommen. Zudem können diese Tools dadurch herausfinden, welche Ressourcen Argo CD selbst verwaltet.

Das annotationsbasierte Ressourcentracking lässt sich in Argo CD üblicherweise in der ConfigMap-Datei argocd-cm konfigurieren, und zwar im Feld

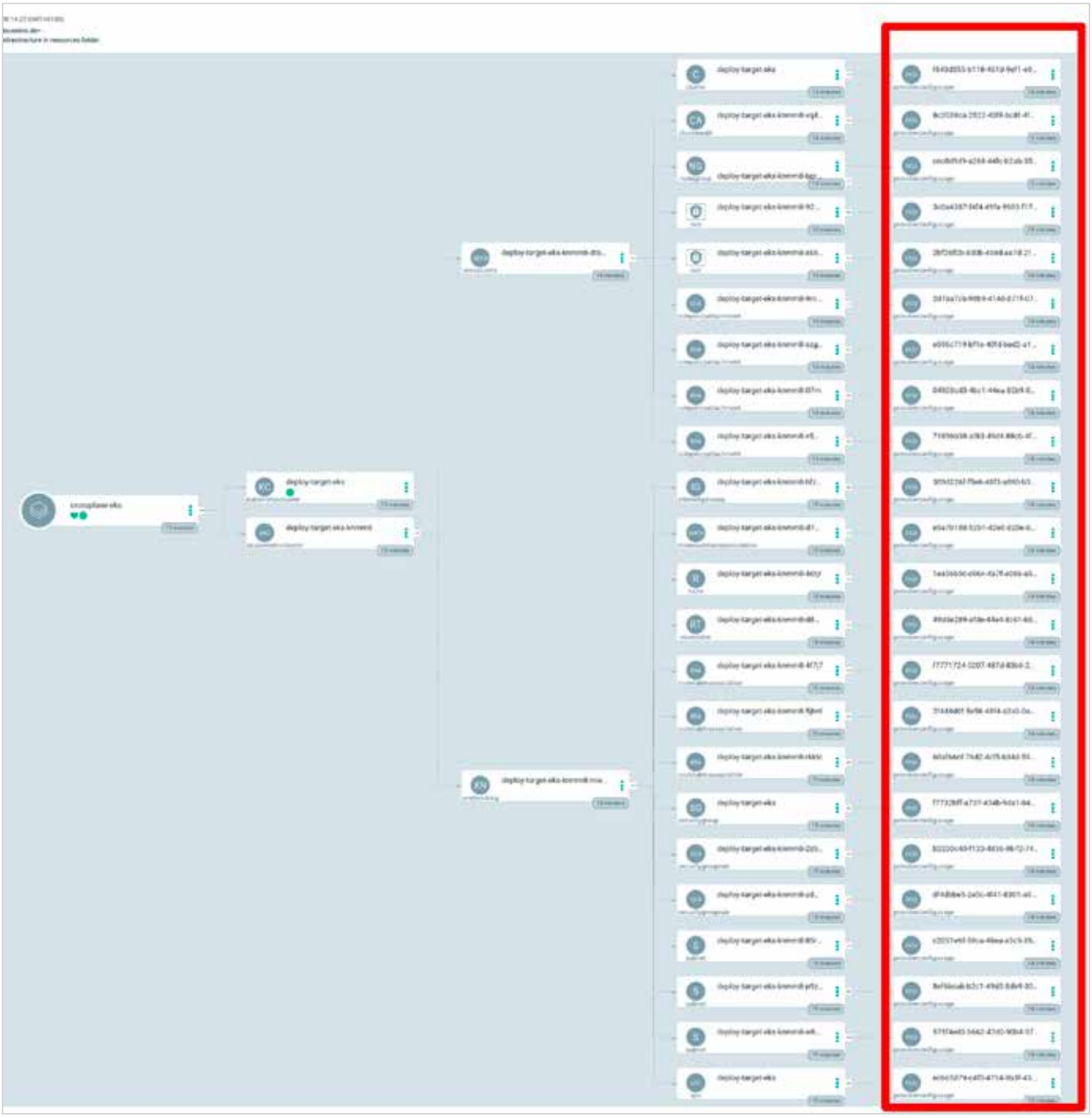
```
application.resourceTracking↵
                    Method: annotation
```

In diesem Fall wird die Konfiguration aber nicht händisch eingetragen, sondern über die Patchdatei argocd-cm-patch.yaml in die Argo-CD-Installation eingebracht.

Ausschluss der Crossplane-generierten CRDs

Die zweite notwendige Konfiguration vor der eigentlichen Argo-CD-Installation betrifft die Exklusion der von Crossplane generierten ProviderConfigUsage-CRDs. Denn die jeweiligen Crossplane-Provider generieren für jede Managed Resource (MR) eine CRD `ProviderConfigUsage`, die sie mitbringen. Sie repräsentiert die Beziehung zwischen der MR und einer ProviderConfig und dient dem Crossplane-Operator als `finalizer`, sollte die ProviderConfig gelöscht werden. Allerdings sind diese ProviderConfigUsage-CRDs explizit nicht dafür gedacht, dass ein Nutzer von Crossplane mit diesen interagiert.

Dennoch würde ohne explizite Konfiguration jede Crossplane-Ressource im Grunde doppelt in Argo CD auftauchen, und zwar als die eigentliche Crossplane-Ressource und als deren ProviderConfig-



Hat man die Exklusion der ProviderConfigUsage-CRDs noch nicht konfiguriert, tauchen alle Crossplane-Ressourcen in Argo CD doppelt auf (Abb. 2).

Usage-CRD (siehe Abbildung 2). Das führt schnell zu einer sehr trägen Argo-CD-Oberfläche.

Da man aber nicht mit diesen CRDs arbeitet und ihre Ansicht keinerlei Mehrwert bietet, kann man sie einfach entfernen. Dafür konfiguriert man üblicherweise im Feld `resource.exclusions` in der ConfigMap argocd-cm die Exklusion der ProviderConfigUsage-CRDs (siehe Listing 3).

Diese Exklusion der Crossplane-generierten ProviderConfigUsage-CRDs wird wie das annotationsbasierte Ressourcentracking über den Patch-File argocd-cm-patch.yaml in die Argo-CD-Konfiguration eingebracht (siehe Listing 4).

Nun sollten alle Dateien für die Installation von Argo CD mithilfe von Kustomize vorbereitet sein. Die Installation in den bereits vorbereiteten kind-Cluster startet man nun im root-Verzeichnis des Projekts mit dem Kommando

```
kubectl apply -k argocd/install
```

Um Folgefehler zu vermeiden, sollte man sicherstellen, dass alle Argo-CD-Komponenten erfolgreich in kind installiert sind. Ist das der Fall, liefert der Befehl

```
kubectl wait --for=condition=ready ↵
          pod -l app.kubernetes.io/↵
     name=argocd-server --namespace↵
               argocd --timeout=300s
```

die Rückgabe `condition met`.

Das Argo-CD-UI aufrufen

Einer der Gründe, Argo CD als GitOps-Werkzeug einzusetzen, ist sicherlich die mitgelieferte Oberfläche, die auch weniger Kubernetes-affinen Nutzern wertvolle Einblicke bietet. Doch selbst Profis profitieren vom sehr guten Überblick über alle Komponenten im Cluster. Für ihren Aufruf bedarf es allerdings eines Passworts. Die Installationsroutine legt ein Initialpasswort für den User `admin` an, das sich mit dem folgenden Befehl auslesen lässt:

```
kubectl -n argocd get secret argocd-↵
   initial-admin-secret -o jsonpath=↵
 "{.data.password}" | base64 -d; echo
```

Des Weiteren ist dafür Sorge zu tragen, dass der Service `argocd-server` von außerhalb des Managementclusters zugreifbar ist. Dafür gibt es eine Vielzahl von Möglichkeiten. Die einfachste ist ein Port-Forwarding, das mit dem Befehl

```
kubectl port-forward -n argocd ↵
        --address='0.0.0.0' service/↵
                argocd-server 8080:80
```

vom clusterinternen Port 80 auf den Host-Port 8080 gemappt wird. Damit sollte die Argo-CD-Oberfläche im Browser unter http://localhost:8080 aufrufbar sein. Nachdem man sich mit dem Benutzernamen admin und dem oben ausgelesenen Passwort angemeldet hat, sollte man vor allem in einem produktiven Set-up das Initialpasswort ändern.

Ein lokales Helm Chart für Crossplane

Auch für die Installation von Crossplane gibt es mehrere Optionen. Allerdings unterschlägt die Crossplane-Dokumentation eine Möglichkeit, wie die Installation auch kompatibel mit Renovate funktionieren könnte. Da bereits die Argo-CD-Installation entsprechend kompatibel konfiguriert wurde und automatisch durch Renovate aktuell gehalten wird, sollte dasselbe auch für die Crossplane-Installation gelten.

Da Renovate irgendeine Art von Versions-gepinnter Dependency-Datei benötigt, bietet sich hier ein lokales Helm Chart an, um die Crossplane-Version zu pinnen. Dadurch kann Renovate die Updates übernehmen. Zudem arbeitet Argo CD wunderbar mit Helm zusammen, sodass die Crossplane-Installation perfekt integriert werden kann.Für das lokale Helm Chart sollte man im Projektverzeichnis des Repositorys einen neuen Ordner namens `crossplane` erstellen. Darin findet das lokale Helm Chart mit dem Namen Chart.yaml Platz (siehe Listing 5).

Für ein vollständig automatisiertes Bootstrapping sollte Argo CD in der Lage sein, dieses Helm Chart ohne manuelle Konfiguration über die Oberfläche zu nutzen. Da das lokale Helm Chart kein Standard-Chart ist, benötigt es ein Secret, das den Ort des eigentlichen Crossplane-Helm-Charts definiert.

Da das vorliegende Set-up schrittweise erweitert werden soll, empfiehlt es sich, für das Secret im Verzeichnis argocd einen neuen Ordner anzulegen, beispielsweise argocd/crossplane-bootstrap. In ihn kommt die Datei crossplane-helm-secret.yaml mit dem Manifest für das Secret (siehe Listing 6).

Listing 6: crossplane-helm-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: crossplane-helm-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  name: crossplane
  url: https://charts.crossplane.io/stable
  type: helm
```

Listing 7: crossplane.yaml

```yaml
# The Argo CD Application for crossplane core components themselves
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: crossplane
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://github.com/jonashackt/crossplane-argocd
    targetRevision: HEAD
    path: crossplane
  destination:
    server: https://kubernetes.default.svc
    namespace: crossplane-system
  syncPolicy:
    automated:
      prune: true
    syncOptions:
    - CreateNamespace=true
    retry:
      limit: 1
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 1m
```

Das Secret sollte dem Managementcluster vor der eigentlichen Crossplane-Installation mit dem Befehl

```
kubectl apply -f argocd/crossplane-↵
         bootstrap/crossplane-helm-↵
                        secret.yaml
```

bekannt gemacht werden.

Ohne Argo-CD-Application läuft nichts

Nun kann Argo CD die Crossplane-Installation übernehmen. Das bedingt die Anlage einer Argo-CD-Application. Auf diese Weise kann man Argo CD deklarativ mitteilen, dass es das zuvor angelegte Helm Chart für die Crossplane-Installation zu nutzen hat. Das Argo-CD-Application-Manifest legt man in der Datei crossplane.yaml im bereits erstellten Ordner argocd/crossplane-bootstrap ab (siehe Listing 7).

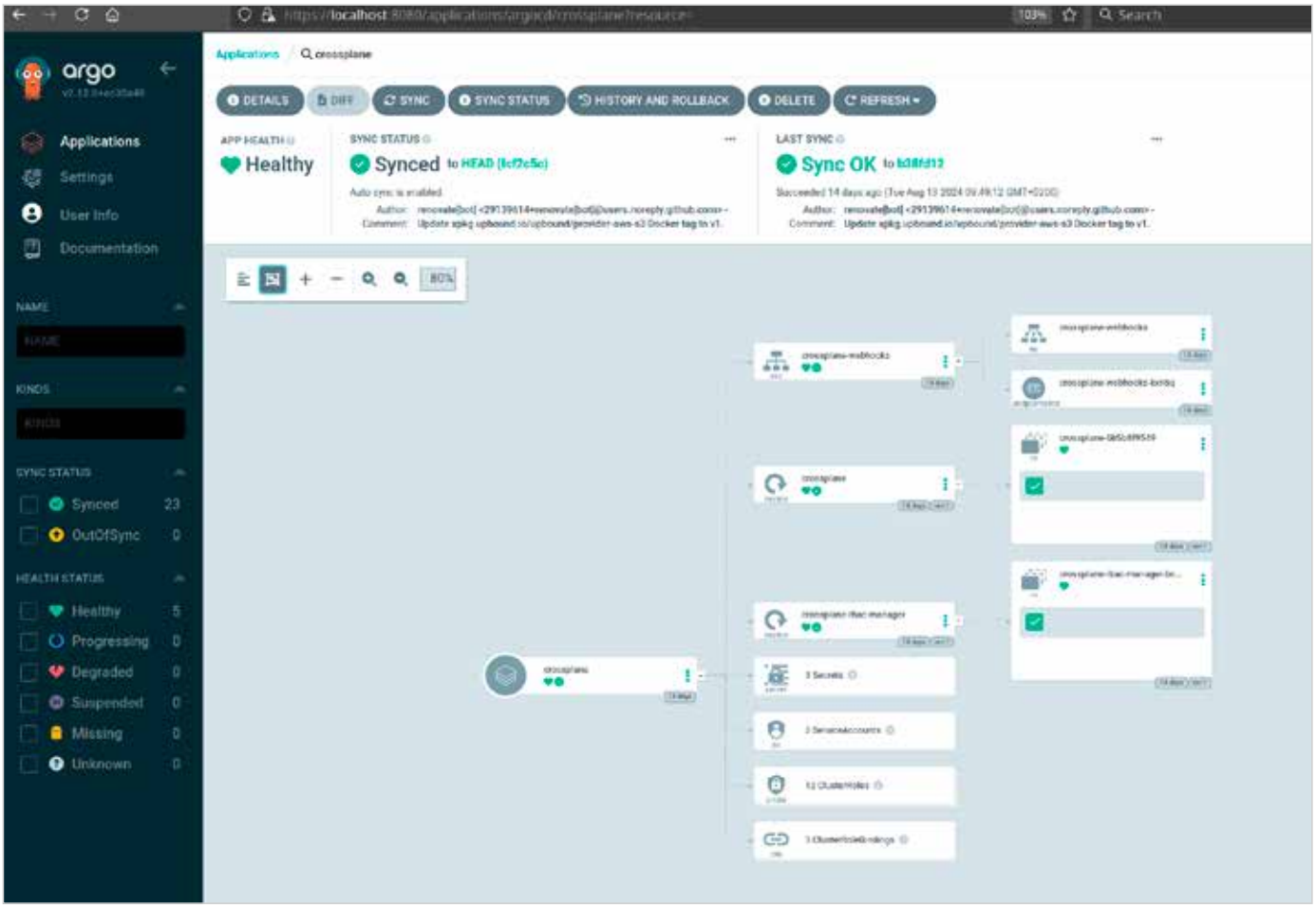
Im Application-Manifest gibt es einiges zu beachten. Zuerst fällt der spezielle `finalizer` auf. Denn ein Löschen des Application-Manifests würde nicht die dadurch deployten Crossplane-Komponenten entfernen. Ein Befehl wie

```
kubectl delete -n argocd -f argocd/↵
 crossplane-bootstrap/crossplane.yaml
```

würde also die Argo-CD-Application löschen und diese auch aus der Oberfläche entfernen, die eigentliche Crossplane-Installation bliebe aber davon unberührt. Da die Argo-CD-Application das gesamte Lifecycle-Management – also sowohl die Installation wie auch die Deinstallation – von Crossplane übernehmen soll, ist hier ein kaskadierendes Löschen notwendig. Damit Argo CD auch die durch die Application angelegten Komponenten löscht, muss der `finalizer` den Wert `resources-finalizer.argocd.argoproj.io` erhalten.

Argo CD übernimmt die Crossplane-Installation

Neben dem `finalizer` ist die Definition des Feldes `spec.source` von Bedeutung. Es legt das Git-Repository und den darin befindlichen Pfad namens crossplane fest, in dem das zuvor definierte Helm Chart zu finden ist. Dazu dienen die Parameter `repoURL` und `path`. Dadurch wird Argo CD ganz in GitOps-Manier auf das Helm Chart konfiguriert und deployt die dahinterliegende Anwendung ebenso wie jegliche Änderung daran, die in Git eingecheckt wird.



Argo CD installiert Crossplane (Abb. 3).

Das Feld `destination` legt fest, auf welchen Kubernetes-Cluster Argo CD die in `spec.source` definierte Anwendung installieren soll. Da Crossplane auf dem lokalen kind-Cluster laufen soll, ist dieser lokale Server mit `https://kubernetes.default.svc` anzugeben. Der für die Installation von Crossplane notwendige Namespace `crossplane-system` ist dann unter `namespace` auch direkt mit konfiguriert.

Als Letztes legt `syncPolicy` Details zur Art und Weise des Crossplane-Deployments fest. Der Parameter `syncPolicy: automated` gibt die vollständige automatisierte Synchronisation durch Argo CD vor. Da der GitOps-Operator von Argo CD das Deployment vollständig übernimmt, ist auch kein Zugriff einer CI/CD-Pipeline mehr auf den Cluster nötig. Damit setzt Argo CD das GitOps-Pull-Prinzip bereits bei der Crossplane-Installation um.

Putzdienst und Fehlerbehandlung einrichten

Zudem wird dem Parameter `syncPolicy: automated` mit `prune: true` noch ein weiterer Parameter hinzugefügt. Er weist Argo CD an, automatisch Crossplane-Ressourcen zu entfernen, die nicht mehr in Git definiert sind.

Dank des Parameters `syncOptions: - CreateNamespace=true` wird Argo CD den für die Crossplane-Installation notwendigen Namespace `crossplane-system` automatisch anlegen. Zuletzt definiert der `retry`-Parameter, was bei Fehlern während des Crossplane-Deployments durch Argo CD zu geschehen hat.

Damit ist es an der Zeit, Argo CD die Crossplane-Installation tatsächlich durchführen zu lassen. Der Befehl

```
kubectl apply -n argocd -f argocd/↵
 crossplane-bootstrap/crossplane.yaml
```

macht die `Application` dem Managementcluster bekannt und löst damit die Crossplane-Installation aus. Nun sollte Argo CD die Crossplane-Kernkomponenten ausrollen. Nachverfolgen lässt sich das auf der Argo-CD-Oberfläche: Die Crossplane-Installation besteht aus dem Operator und dem RBAC-Manager, einem Webhooks-Service und Secrets, ServiceAccounts und ClusterRoles (siehe Abbildung 3).

Ausblick

Damit ist die Grundinstallation von Crossplane mithilfe von Argo CD abgeschlossen. Das komplette Set-up ist automatisiert in einer CI/CD-Pipeline bereitstellbar und kann durch Renovate aktuell gehalten werden.

Der letzte Teil des Tutorials erweitert die Installation um konkrete Crossplane-Provider und macht sie somit fit für die eigentliche Infrastrukturprovisionierung. Dabei kommen fortgeschrittene Konzepte in Argo CD wie das App-of-Apps-Pattern und SyncWaves zum Einsatz. (sun@ix.de)

Quellen

[1] Jonas Hecht; Tutorial GitOps mit Crossplane, Teil 1: Beyond CI/CD; iX 5/2025, S. 122

[2] Alle Ressourcen siehe ix.de/z17z.

JONAS HECHT

ist Senior Solution Architect bei codecentric.