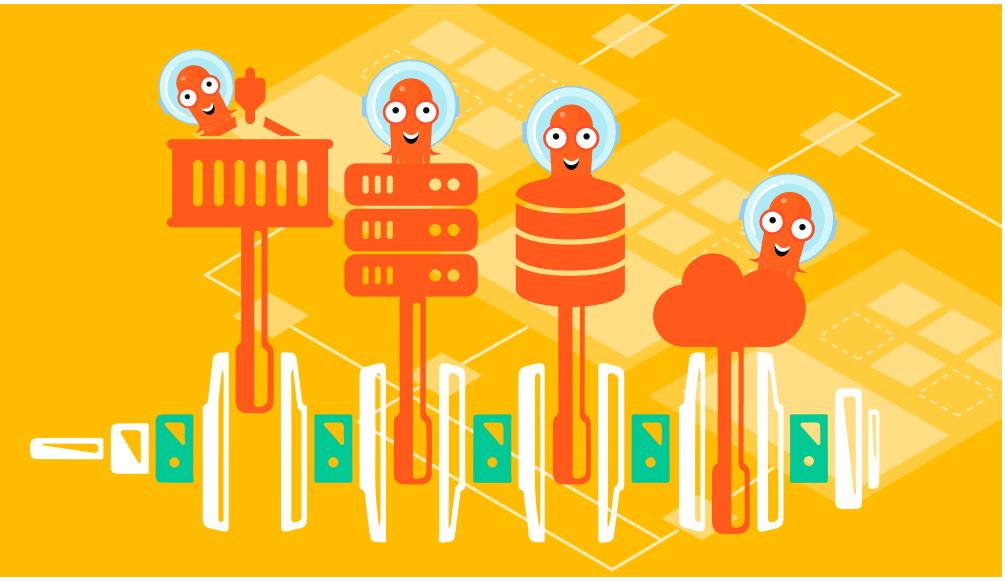


GitOps mit Crossplane, Teil 3: Provider

Nachdem der Provisionierer Crossplane und das GitOps-Werkzeug Argo CD selbst nach den GitOps-Regeln eingerichtet wurden, stellen zwei Argo-CD-Features den fehlerfreien Ablauf des Deployments sicher: SyncWaves und App-of-Apps-Pattern. Derweil übernehmen die Crossplane-Provider das eigentliche Einrichten der Infrastruktur.

Von Jonas Hecht

Der zweite Teil des Tutorials zeigte, wie Crossplane als Argo-CD-Application zu konfigurieren ist und wie die Basisinstallation beider Werkzeuge vonstattengeht. Im vorliegenden dritten und letzten Teil kommen zusätzlich die Crossplane-Provider und ihre Konfigurationen ins Spiel. Fortgeschrittene Argo-CD-Konzepte wie SyncWaves und App-of-Apps-Pattern helfen dabei, das Set-up zu vervollständigen und auf die eigentliche Infrastrukturprovisionierung in der AWS-Cloud mit Crossplane und Argo CD vorzubereiten.

X-TRACT

- Nachdem im zweiten Teil des Tutorials die Basisinstallation von Argo CD und Crossplane abgeschlossen wurde, geht es im dritten und letzten Teil um die Crossplane-Provider und ihre Konfigurationen.
- Die Crossplane-Provider übernehmen nach dem Abschluss der Vorbereitungen das eigentliche Provisionieren der Infrastruktur.
- Erst im Zusammenspiel der Argo-CD-Features App-of-Apps-Pattern und SyncWaves lässt sich die für einen fehlerfreien Ablauf benötigte Reihenfolge des Deployments eindeutig definieren.
- Im beschriebenen Set-up halten Argo CD und Crossplane nicht nur die von ihnen ausgerollte Infrastruktur aktuell, sondern auch sich selbst.

Tutorialinhalt

Alle Schritte in diesem Tutorialteil setzen die Vorarbeiten aus dem zweiten Teil voraus, einschließlich der dort beschriebenen Kommandozeilenwerkzeuge und des lokalen kind-Clusters [1, 2]. Um das Set-up für die eigentliche Infrastrukturprovisionierung in der Cloud fit zu machen, braucht es einen Crossplane-Provider. Abbildung 1 gibt einen Überblick über die Komponenten, die im Laufe dieses Artikels Verwendung finden. Alle Codebeispiele sind wie bei den vorangegangenen Teilen auf GitHub abruf- und nachvollziehbar (alle Ressourcen siehe ix.de/zcrr).

Ohne Secret kein Zugriff auf die Cloud

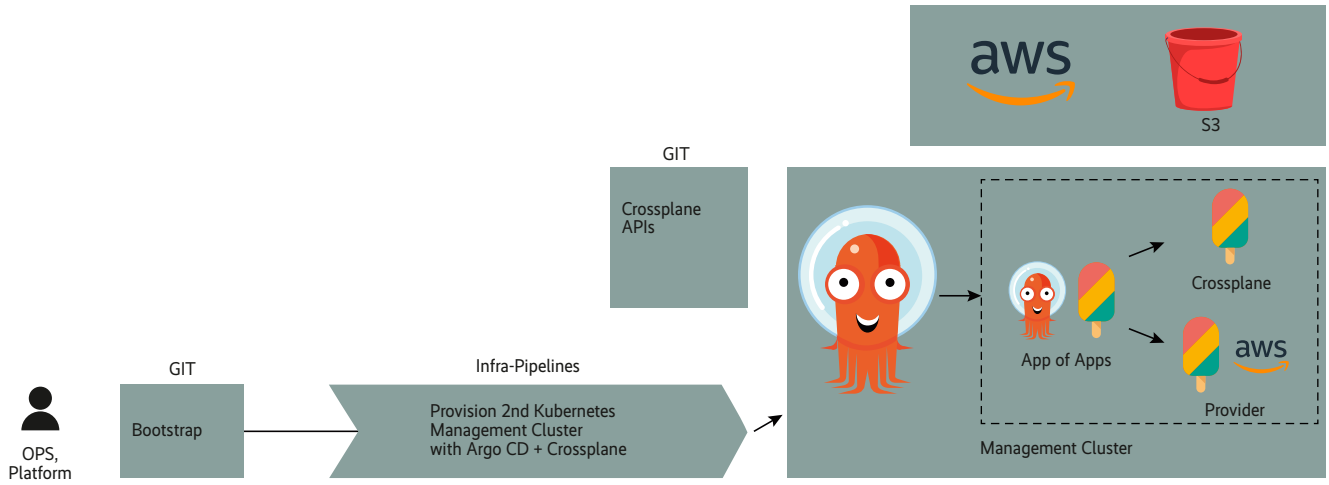
Erst die Crossplane-Provider bringen die einzelnen Managed Resources (MRs) mit, die den Zugriff auf die Cloud-Provider erlauben. In der Regel müssen die Crossplane-Provider aber erst befähigt werden, mit der jeweiligen Cloud zu kommunizieren. Da der Artikel beispielhaft die AWS-Cloud nutzt, müssen auch die entsprechenden AWS-Credentials bereitstehen. Dafür ist die lokale Installation der AWS-CLI Voraussetzung.

Die IAM-Credentials (Identity and Access Management) lassen sich Crossplane mithilfe eines Secrets mitteilen. Zum Anlegen des Secrets ist eine Datei aws-creds.conf im root-Verzeichnis des Projekts anzulegen (siehe Listing 1). Wichtig: Diese Datei enthält Credentials und gehört nicht in das Versionskontrollsystem. Eine .gitignore-Datei mit dem Eintrag `*-creds.conf` verhindert das Einchecken der Credentials-Datei in das Repository.

Mit Rückgriff auf die Datei aws-creds.conf kann nun der Befehl

```
kubectl create secret generic aws-↵
  creds -n crossplane-system --from-↵
          file=creds=./aws-creds.conf
```

das für den Crossplane-Provider benötigte Secret anlegen. Wer seine Secrets zentral in Git verwalten will, kommt nicht um ein Secrets Management herum [3].



Das vollständige Set-up mit Crossplane und Argo CD ist in wenigen Schritten erledigt (Abb. 1).

Er kann beispielsweise statt eines einfachen Kubernetes-Secrets oder Vault in Argo CD auch ein nachgelagertes Secrets Management wie den External Secrets Operator nutzen.

Argo CD installiert den Crossplane-Provider für AWS

Grundsätzlich gibt es mehrere Arten von Crossplane-Providern. Zuletzt haben sich aber die Upbound-Provider-Familien als diejenigen etabliert, die das größte Spektrum an Ressourcen der großen Cloud-Provider abdecken. Deshalb kommt auch hier der Upbound-Provider für AWS zum Einsatz.

Zu diesem Zweck ist zuerst ein neuer Ordner upbound/provider-aws im root-Verzeichnis des Projekts anzulegen. Durch das Etablieren solcher Namenskonventionen wird sichergestellt, dass auch andere Provider ihren Platz im Set-up finden, etwa unter upbound/provider-azure oder crossplane-contrib/provider-argocd – letzterer etwa aus der Sammlung der Community-Contributions-Provider.

In einem Unterordner provider des Verzeichnisses upbound/provider-aws lassen sich nun die Einzelprovider der AWS-Provider-Familie unterbringen, also etwa S3, IAM, EC2 und EKS. Was es mit den Providerfamilien auf sich hat, verrät der Blogpost von Jared Watts auf crossplane.io (siehe ix.de/zcrr). Für das Verständnis des vorliegenden Artikels genügt es zu wissen, dass eine Segmentierung der Provider aufgrund der großen Zahl der CRDs (Custom Resource Definitions) notwendig wurde, die sonst die Kubernetes-API zu stark ausgebremst hätten. Im Verzeichnis provider enthält die zu erstellende Datei upbound-provider-aws-s3.yaml das eigentliche Providermanifest (siehe Listing 2).

Wie für Crossplane selbst ist auch für den Crossplane-Provider das Anlegen einer Argo-CD-Application sinnvoll. Damit ist sichergestellt, dass Argo CD die Installation und das Management des Providers übernimmt. Dazu ist im bereits existierenden Ordner argocd/crossplane-bootstrap eine neue Datei crossplane-provider-aws.yaml anzulegen (siehe Listing 3).

Listing 1: aws-creds.conf

```
echo "[default]
aws_access_key_id = $(aws configure get aws_access_key_id)
aws_secret_access_key = $(aws configure get aws_secret_access_key)
" > aws-creds.conf
```

Listing 2: upbound-provider-aws-s3.yaml

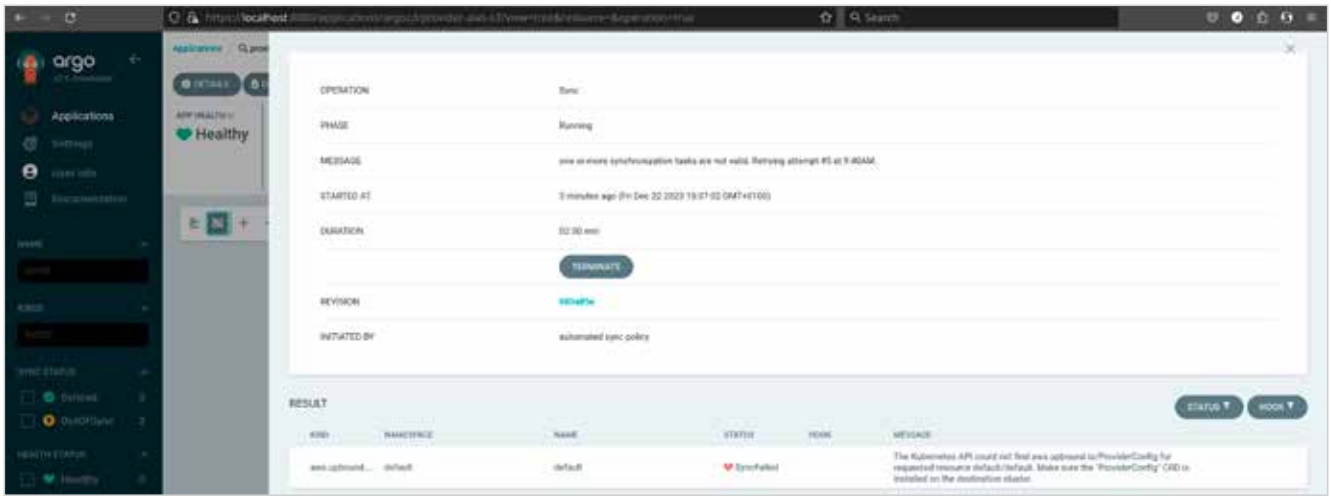
```
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: upbound-provider-aws-s3
spec:
  package: xpkg.upbound.io/upbound/provider-aws-s3:v1.20.0
  packagePullPolicy: Always
  revisionActivationPolicy: Automatic
  revisionHistoryLimit: 1
```

Listing 3: crossplane-provider-aws.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: crossplane-provider-aws-s3
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    path: upbound/provider-aws/provider
    repoURL: https://github.com/jonashackt/crossplane-argocd
    targetRevision: HEAD
  destination:
    namespace: default
    server: https://kubernetes.default.svc
  syncPolicy:
    automated:
      prune: true
```

Argo CD überwacht den vollständigen Lebenszyklus

Wie bei der Argo-CD-Application für Crossplane konfiguriert der Parameter `spec.source.path` das Verzeichnis des Providermanifests innerhalb des Git-Repositorys. Auf diese Weise kann Argo CD das Management des Providers vollständig übernehmen. Auch der im zweiten Artikel beschriebene `finalizer` ist wie-



Der Sync des Crossplane-Providers schlägt fehl (Abb. 2).

der mit an Bord, sodass der Provider wieder abgeräumt wird, sollte die Argo-CD-Application gelöscht werden.

Wichtig ist auch wieder das Flag `syncPolicy.automated`. Es aktiviert die GitOps-Features, sodass Argo CD nun den vollständigen Lebenszyklus der damit installierten Komponenten beaufsichtigt. Ohne diese Konfiguration würde der Crossplane-Provider den Fehler

```
Resource not found in cluster: pkg.↵
          crossplane.io/v1/Provider:↵
              upbound-provider-aws-s3
```

ausgeben. Nun steht alles bereit, sodass Argo CD den Crossplane-Provider auf dem Managementcluster installieren kann. Dafür muss die Argo-CD-Application dem Managementcluster nur noch mit dem Befehl

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

bekannt gemacht werden.

Tatsächlich ist danach sogar mit einem Fehler innerhalb von Argo CD zu rechnen. Er ist in der über http://localhost:8080 erreichbaren Argo-CD-Oberfläche sichtbar, sofern sie wie im zweiten Teil des Tutorials beschrieben eingerichtet wurde (siehe Abbildung 2). Doch er war erwartbar und wird im nächsten Abschnitt behoben.

Ohne die ProviderConfig läuft es nicht

Um den Crossplane-Provider lauffähig zu bekommen, bedarf es der passenden Datei ProviderConfig. Sie verbindet den Crossplane-Provider vor allem mit dem passenden Secret für die Cloud-Provider-Credentials. Hierfür ist im Ordner upbound/provider-aws ein neues Verzeichnis config und darin eine neue Datei namens provider-aws-config.yaml anzulegen (siehe Listing 4).

Innerhalb des Manifests ProviderConfig sind vor allem die Felder `secretRef.name` und `secretRef.key` von Bedeutung, denn sie verweisen auf das zuvor erstellte Secret. Die Crossplane-Provider benutzen die `ProviderConfig` mit dem Namen `default`, wenn kein anderer angegeben wird. Das hat den Charme, dass der AWS-Provider für alle AWS-Ressourcen diese `ProviderConfig` nutzt.

Wie der Crossplane-Provider selbst sollte auch die `ProviderConfig` eine eigene Argo-CD-Application bekommen. Dazu muss eine Datei crossplane-provider-aws-config.yaml im Verzeichnis argocd/crossplane-bootstrap existieren. Im Unterschied zur Application für den Crossplane-Provider zeigt der Parameter `spec.source.path` aber auf den Pfad upbound/provider-aws/config (siehe Listing 5).

Listing 4: provider-aws-config.yaml

```
apiVersion: aws.upbound.io/v1beta1
kind: ProviderConfig
metadata:
  name: default
spec:
  credentials:
    source: Secret
    secretRef:
      namespace: crossplane-system
      name: aws-creds
      key: creds
```

Listing 5: crossplane-provider-aws-config.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: provider-aws-config
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    path: upbound/provider-aws/config
    repoURL: https://github.com/jonashackt/crossplane-argocd
    targetRevision: HEAD
  destination:
    namespace: default
    server: https://kubernetes.default.svc
  syncPolicy:
    automated:
      prune: true
```

Damit sollte tatsächlich alles für ein funktionierendes Deployment des Crossplane-Providers im Managementcluster durch Argo CD bereitstehen. Nun ist nur noch das ProviderConfig-Manifest innerhalb des Managementclusters mit dem Befehl

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

bekannt zu machen. Damit wurde die Basisinstallation um die benötigten Crossplane-Komponenten zur Providerinstallation und -konfiguration ergänzt (siehe Abbildung 3).

Argo-CD-SyncWaves könnten nützlich sein

Obwohl das Set-up aus Crossplane und Argo CD nun vollständig ist, gilt es, noch einen wichtigen Punkt zu optimieren. Denn bereits jetzt ist eine ganze Reihe Argo-CD-basierter Application-Dateien entstanden. Nun sind sie noch in der richtigen Reihenfolge im Managementcluster zu applizieren, damit das korrekte Deployment der Crossplane-Komponenten funktioniert.

Die Situation verschärft sich weiter, wenn neue Erweiterungen wie der External Secrets Operator oder weitere Crossplane-Provider hinzukommen. Hier wäre es wünschenswert, ein zentrales Manifest zu haben, das die Reihenfolge des Crossplane-Set-ups definiert und Argo CD damit einen vollständigen Bauplan liefert.

Listing 6: crossplane-bootstrap.yaml

```
# The Argo CD App of Apps for all Crossplane components
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: crossplane-bootstrap
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://github.com/jonashackt/crossplane-argocd
    targetRevision: HEAD
    path: argocd/crossplane-bootstrap
  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
```

Listing 7: crossplane-helm-secret.yaml

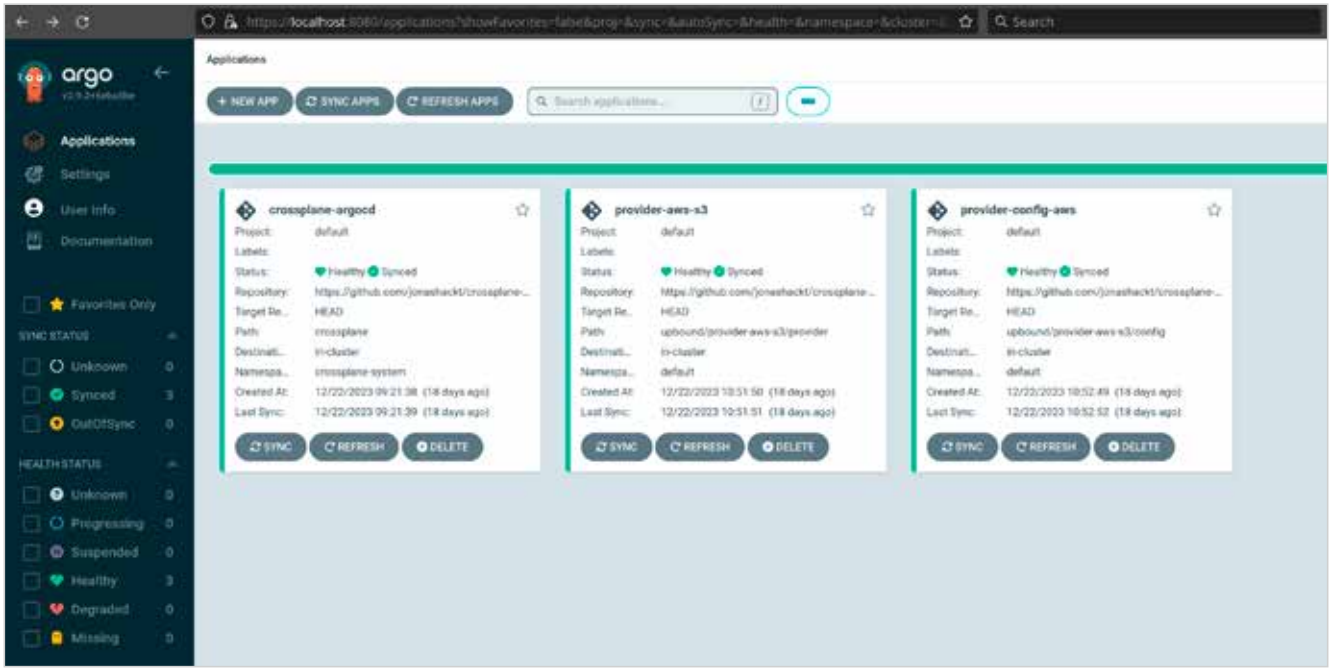
```
metadata:
  annotations:
    argocd.argoproj.io/sync-wave:
"0"
```

Listing 8: crossplane.yaml

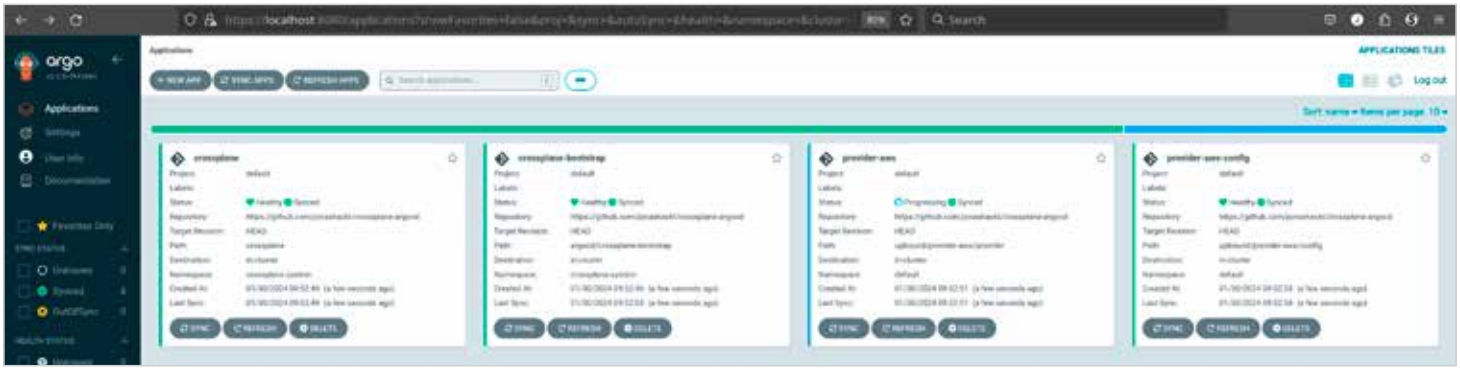
```
metadata:
  annotations:
    argocd.argoproj.io/sync-wave:
"1"
```

Listing 9: Alle bisherigen Manifeste aus dem Managementcluster entfernen

```
kubectl delete -n argocd -f argocd/↵
    crossplane-bootstrap/crossplane-↵
            provider-aws-config.yaml
kubectl delete -n argocd -f argocd/↵
    crossplane-bootstrap/crossplane-↵
                    provider-aws.yaml
kubectl delete -n argocd -f argocd/↵
 crossplane-bootstrap/crossplane.yaml
kubectl delete -n argocd -f argocd/↵
    crossplane-bootstrap/crossplane-↵
                    helm-secret.yaml
```



Alle Crossplane-Komponenten wurden erfolgreich im Managementcluster deployt (Abb. 3).



Mithilfe des App-of-Apps-Patterns wurden alle Crossplane-Komponenten installiert (Abb. 4).

Ein erster naiver Ansatz könnte die Implementierung einer Argo-CD-Application sein, die einfach auf das Verzeichnis argocd/crossplane-bootstrap zeigt, in dem alle bisherigen Application-Manifeste liegen. Doch dieser Versuch endet in dem Fehler:

```
The Kubernetes API could not find ↵
  aws.upbound.io/ProviderConfig for ↵
         requested resource default/↵
              default. Make sure the ↵
  "ProviderConfig" CRD is installed ↵
          on the destination cluster.
```

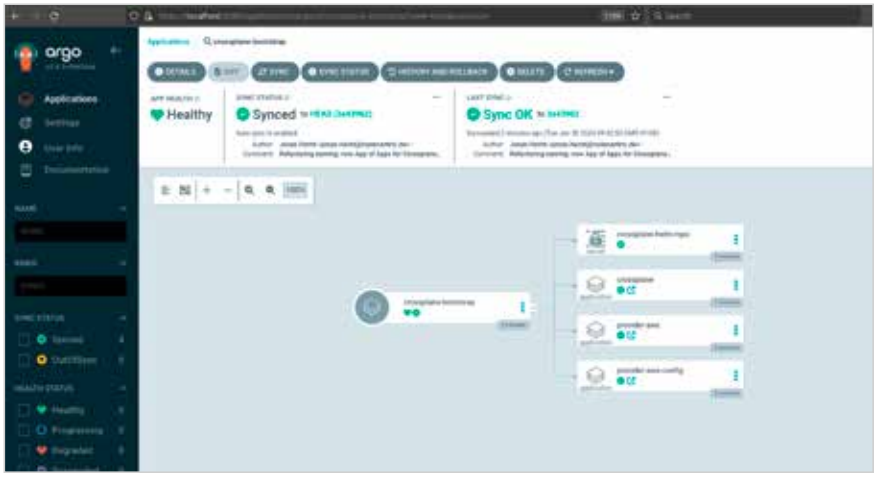
Das Problem der unklaren Deployment-Reihenfolge bleibt damit nämlich weiter bestehen. Beispielsweise muss das Providermanifest vor der `ProviderConfig` deployt werden, sonst schlägt das Deployment aufgrund fehlender CRDs im Cluster fehl.

Theoretisch könnte das Argo-CD-Feature SyncWaves hier helfen. Denn grundsätzlich lässt sich mit seiner Hilfe definieren, welche Kubernetes-Manifeste Argo CD in welcher Reihenfolge deployen soll. Allerdings müsste man ohne weitere Maßnahmen wirklich alle an dem Deployment beteiligten Manifeste über den Parameter `argocd.argoproj.io/sync-wave` mit einer SyncWaves-Konfiguration versehen. Dies hieße aber, alle Manifeste für die Crossplane-Installation anzupassen, was mit einem extrem hohen Aufwand einherginge.

Das App-of-Apps-Pattern hilft aus der Patsche

Es gibt allerdings ein weiteres Argo-CD-Feature, das in Kombination mit den SyncWaves genau diese Klimmzüge verhindern hilft. Das App-of-Apps-Pattern liefert exakt das fehlende Puzzlestück, mit dem man das gesamte Crossplane-Set-up mit einem einzelnen Manifest strukturieren kann. Denn mit dem Pattern lassen sich Argo-CD-Applications innerhalb von Argo-CD-Applications nutzen, da sie ja selbst auch nur Kubernetes-Manifeste sind.

Das bedeutet, man verweist in einer Art Hauptmanifest auf die verschiedenen Teilmanifeste. In ihnen lässt sich dann das SyncWaves-Feature so anwenden, dass es das Deployment in der richtigen Reihenfolge vorgibt. Bei dieser Vorgehensweise müssen keine Installationsmanifeste von Crossplane angepasst werden.



In der Detailansicht App-of-Apps-Application liefert Argo CD alle Details zum Deployment (Abb. 5).

Um das App-of-Apps-Pattern zu nutzen, ist zuerst das Hauptmanifest anzulegen. Es bildet später den Startpunkt für das gesamte Crossplane-Set-up. Dieses Hauptmanifest kann beispielsweise in der Datei crossplane-bootstrap.yaml im Ordner `argocd` liegen (siehe Listing 6).

Das Manifest weist diese Argo-CD-Application an, im Pfad argocd/crossplane-bootstrap des Git-Repositorys, in dem sich alle bisher angelegten Application-Manifeste befinden, nach Manifesten zu suchen. Auch im Hauptmanifest ist wieder ein `finalizer` zu definieren, da Argo CD sonst den Lebenszyklus der Crossplane-Komponenten nicht vollständig managen kann.

SyncWaves mit dem App-of-Apps-Pattern kombinieren

Mit dem fertigen Hauptmanifest kann man nun das SyncWaves-Feature in den Teilmanifesten nutzen. Dazu sind alle Manifeste innerhalb des Ordners argocd/crossplane-bootstrap schrittweise anzupassen. Das sollte genau in der Reihenfolge geschehen, in der Argo CD die Crossplane-Komponenten deployen soll.

Zuerst ist also das Secret anzupassen, das das Crossplane-Helm-Chart für Argo CD nutzbar macht. Das Secret wurde bereits im zweiten Tutorialteil in der Datei argocd/crossplane-bootstrap/crossplane-helm-secret.yaml deklariert und ist für das SyncWaves-Feature mit der `metadata`-Definition `annotations: argocd.argoproj.io/sync-wave` zu versehen (siehe Listing 7).

Die Nummerierung der SyncWaves steht dem Anwender frei, sogar negative Werte sind erlaubt. Für eine bessere Lesbarkeit gilt hier die Definition `sync-wave: "0"`, die die früheste Phase des Crossplane-Deployments festlegt.

Dem Secret für das Crossplane-Helm-Chart folgt die Argo-CD-Application, die das Deployment der Crossplane-Kernkomponenten triggert. Sie ist in der Datei

argocd/crossplane-bootstrap/crossplane.yaml beschrieben und wird um die Definition der nächsten SyncWave erweitert (siehe Listing 8).

Crossplane in der richtigen Reihenfolge neu installieren

Dieses Vorgehen zieht sich nun durch alle Anwendungsmanifeste. Der Crossplane-Provider ist als Nächstes an der Reihe und erhält in der Datei argocd/crossplane-bootstrap/crossplane-provider-aws.yaml die Definition `sync-wave: "2"`. Darauf folgt die `ProviderConfig` in der Datei argocd/crossplane-bootstrap/crossplane-provider-config-aws.yaml mit der Definition `sync-wave: "3"`.

Wer auf diese Weise das Crossplane-Deployment in die richtige Reihenfolge gebracht hat, kann nun das App-of-Apps-Pattern zusammen mit den SyncWaves ausprobieren. Wer aber jeden Schritt bis hierhin nachvollzogen hat, sollte zuerst das bisherige Crossplane-Deployment löschen, denn das App-of-Apps-Pattern würde sich damit empfindlich beißen. Dazu kann man auf der Kommandozeile alle bisherigen Manifeste in umgekehrter Reihenfolge aus dem Managementcluster entfernen (siehe Listing 9).

Damit steht der Nutzung des App-of-Apps-Patterns nichts mehr im Wege. Der Befehl

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

nutzt das zuvor definierte Hauptmanifest und weist Argo CD an, alle notwendigen Manifeste in der richtigen Reihenfolge zu deployen (siehe Abbildung 4). Diese Vorgehensweise reduziert die Komplexität des vollständigen Crossplane-Set-up deutlich. Öffnet man in der Argo-CD-Oberfläche die im Hauptmanifest definierte App-of-Apps-Application `crossplane-bootstrap`, offenbart sie alle Crossplane-Komponenten und präsentiert sie übersichtlich (siehe Abbildung 5).

Listing 10: simple-bucket.yaml

```yaml
apiVersion: s3.aws.upbound.io/v1beta1
kind: Bucket
metadata:
  name: crossplane-argocd-simple-bucket
spec:
  forProvider:
    region: eu-central-1
  providerConfigRef:
    name: default
```

Listing 11: aws-s3.yaml

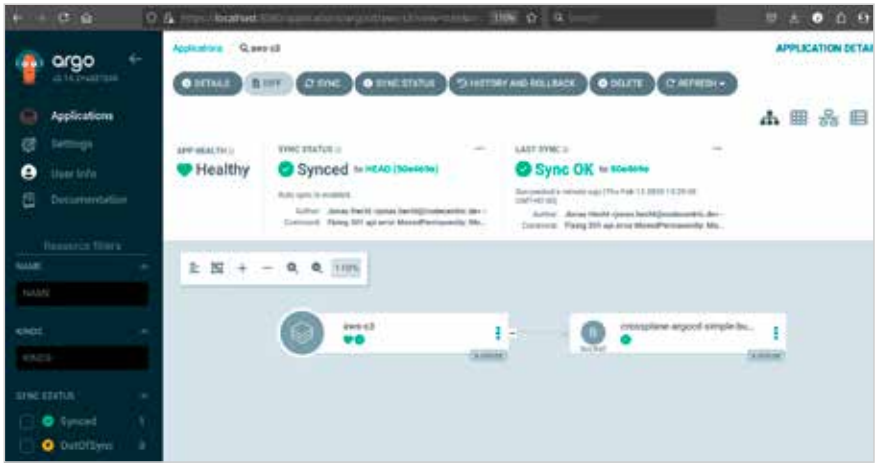
```yaml
# The Argo CD Application for Crossplane Managed Resources
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: aws-s3
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://github.com/jonashackt/crossplane-argocd
    targetRevision: HEAD
    path: infrastructure/s3
  destination:
    namespace: default
    server: https://kubernetes.default.svc
  syncPolicy:
    automated:
      prune: true
    retry:
      limit: 5
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 1m
```

Echte Ressourcen in der AWS-Cloud provisionieren

Offen bleibt allerdings die Frage, ob dieses Set-up tatsächlich eine Cloud-Infrastruktur provisionieren kann. Als möglichst einfaches Beispiel einer Infrastrukturprovisionierung in der AWS-Cloud bietet sich ein S3-Bucket an. Komplexere Infrastrukturen folgen denselben Prinzipien.

Der einfachste Weg, mit Crossplane Cloud-Ressourcen zu provisionieren, besteht darin, die Managed Resources ohne Compositions zu nutzen [4]. Als Startpunkt für das Bucket-Manifest bietet sich die Dokumentation des Crossplane-Providers für AWS S3 an. Sie liefert im Reiter Examples zudem Codebeispiele.

Will man die zu provisionierende Infrastruktur im Projektverzeichnis vom Deployment der Crossplane-Komponen-



Argo-CD-Application triggert die Infrastrukturprovisionierung (Abb. 6).

ten trennen, empfiehlt sich ein neues Verzeichnis namens infrastructure/s3, in dem die Manifestdatei simple-bucket.yaml ihren Platz findet (siehe Listing 10). Die für das S3-Bucket relevante Konfigurationsoption ist die AWS-Region, die mit `eu-central-1` parametrisiert ist.

An dieser Stelle soll Argo CD das Deployment des Bucket mithilfe von Crossplane triggern und den bidirektionalen Reconciliation Loop implementieren, den der erste Teil des Tutorials ausführlich behandelt hat [1]. Denn Argo CD kommt die Rolle zu, Änderungen an der Manifestdatei dem Managementcluster bekannt zu machen, während Crossplane die geänderten Ressourcen provisioniert – so wie sie in Git eingecheckt wurden. Deshalb benötigt man auch an dieser Stelle eine Argo-CD-Application.

Argo-CD-Applications für die Infrastruktur

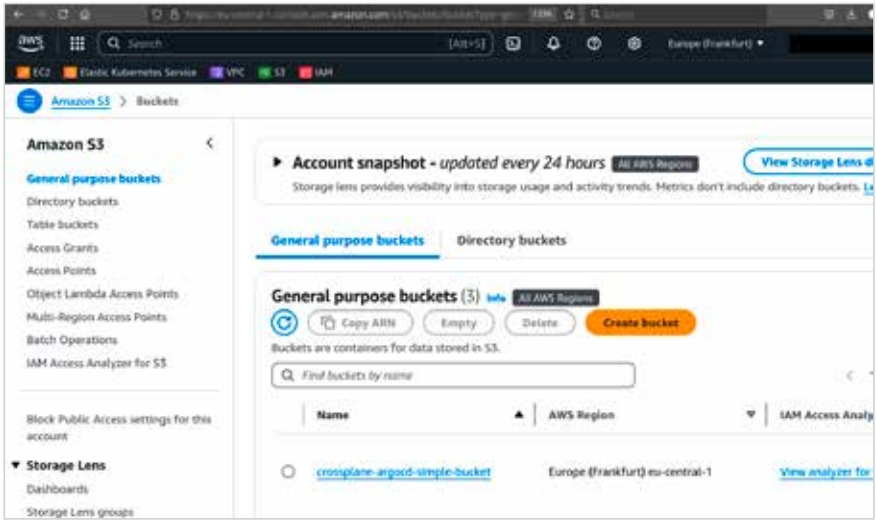
Zu diesem Zweck sollte man ein weiteres Verzeichnis infrastructure im Ordner argocd anlegen, also getrennt vom eigentlichen Crossplane-Deployment. In einem produktiven Set-up würde dieses Manifest in einem eigenen Repository liegen. Der Einfachheit halber liegt es hier lediglich in einem separaten Ordner.

Im neu erstellten Ordner argocd/infrastructure ist dann die Argo-CD-Application anzulegen. Die Manifestdatei heißt aws-s3.yaml (siehe Listing 11).

Die generelle Struktur der Application sollte bereits von den anderen Deployments bekannt sein. Doch dieses Mal sollte das Applizieren der Datei im Managementcluster Argo CD dazu bringen, das Bereitstellen des S3-Bucket über Crossplane zu triggern. Dazu ist dem Managementcluster das Manifest einmalig mit dem Befehl

```
kubectl apply -f argocd/↵
          infrastructure/aws-s3.yaml
```

bekannt zu machen. Ist die Provisionierung erfolgreich verlaufen, sollte die neue Argo-CD-Application in den Status `Healthy` wechseln (siehe Abbildung 6).



Das durch Argo CD und Crossplane provisionierte S3-Bucket wird in der AWS-Konsole sichtbar (Abb. 7).

Auch innerhalb der AWS-Konsole sollte das provisionierte S3-Bucket nun auffindbar sein (siehe Abbildung 7).

Ausblick

Das hier gezeigte vollständig automatisierte Set-up implementiert bereits alle GitOps-Prinzipien und liefert eine stabile Basis für die eigene Platform-Engineering-Initiative. Denn der bidirektionale Reconciliation Loop gewährleistet, dass Argo CD und Crossplane jede in Git definierte Infrastruktur einschließlich aller Updates selbstständig ausrollen. Damit sind die GitOps-Prinzipien auch für die Infrastrukturprovisionierung vollständig implementiert. Nebenbei entfällt das Handling mehrfach vorgehaltener States, wie es bei Terraform/OpenTofu oder Pulumi der Fall ist.

Auch auf zukünftige Erweiterungen ist das Set-up vorbereitet, denn das App-of-Apps-Pattern stellt zusammen mit Sync-Waves die Wartbarkeit und Übersichtlichkeit sicher. Gleichgültig, ob weitere Crossplane-Provider oder Werkzeuge wie der External Secrets Operator integriert werden: Es existiert jederzeit ein vollständiger und aktueller Bauplan der GitOps-Architektur. Zu guter Letzt lässt sich das Set-up mit Renovate automatisch aktuell halten und auf viele weitere Managementcluster ausrollen. (sun@ix.de)

Quellen

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

[2] Jonas Hecht; Tutorial GitOps mit Crossplane, Teil 2: Basisinstallation, iX 6/2025, S. 102

[3] Jan Bundesmann, Pascal Fries, Lukas Paluch; Secrets Management für GitOps; iX 9/2023, S. 66

[4] Jonas Hecht; Crossplane: GitOps für die Multi-Cloud; iX 2/2023, S. 114

[5] Alle Ressourcen finden sich unter ix.de/zcrr.

JONAS HECHT

ist Senior Solution Architect bei codecentric.