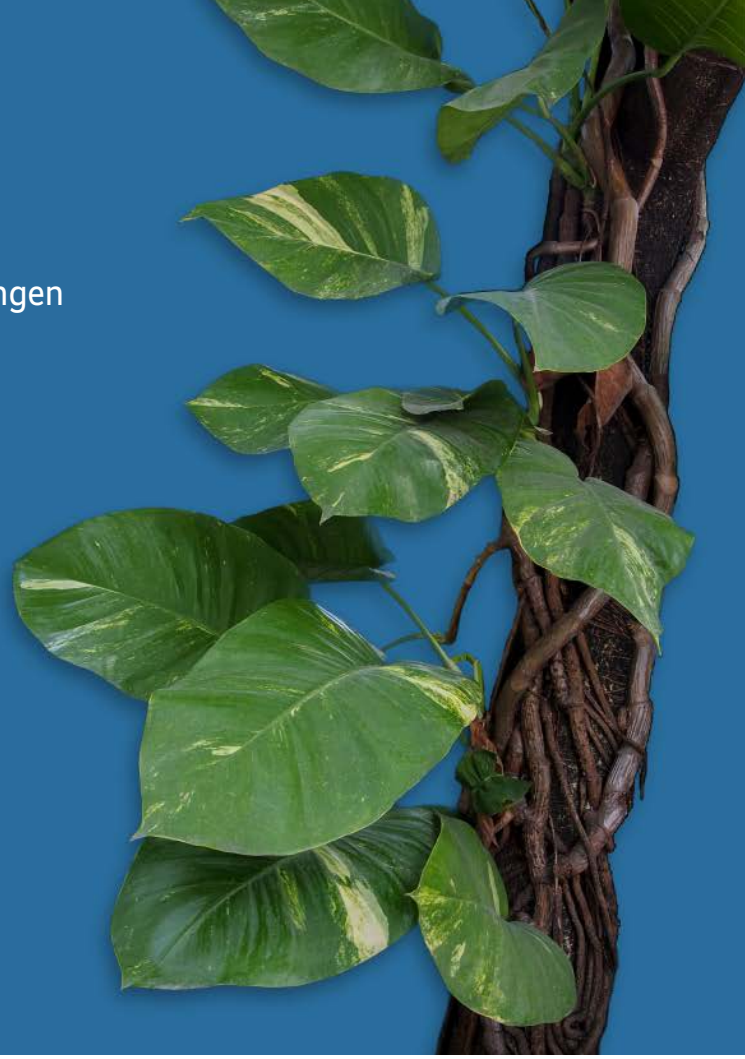


Kubernetes Federation in Multi-Cloud-Umgebungen

Wolkenlenker

Sebastian Meyer, Michel Raabe, André Steincke

Ein Kubernetes-Cluster über Cloud-Grenzen hinweg verspricht auch Ausfälle ganzer Cloud-Regionen abzufangen. Ein Blick auf erschiedene Methoden, das praktisch umzusetzen.

Eine der häufigsten Anforderungen an den Betrieb einer Anwendung im Unternehmen ist Hochverfügbarkeit. In den vergangenen Jahren haben sich viele Konzepte herausgebildet, die den Zugriff auf eine Applikation von einer fehlerhaften Instanz auf eine funktionierende umschwenken. Zeitgleich hat an vielen Stellen ein Umzug von Diensten in die Cloud begonnen, sei es, um Kosten zu sparen oder weil man über die verschiedenen Regionen und Zonen der Cloud auch im Falle von Netzwerkproblemen in einem Datacenter erreichbar sein möchte.

Parallel dazu haben sich für einfacheres Deployment Container-Lösungen wie Kubernetes verbreitet. Sie versprechen hohe Verfügbarkeit etwa durch automatisches Neustarten von Anwendungen oder Load Balancing über mehrere Instanzen hinweg. Allerdings schützt der Einsatz von Kubernetes allein nicht vor dem Ausfall eines ganzen Rechenzentrums oder gar eines Cloud-Anbieters. Multi-Cloud, also der Betrieb mehrerer Kubernetes-Cluster verteilt über verschiedene Regionen beziehungsweise Anbieter, ist ein Ansatz, dieser Problematik beizukommen.

Kubernetes Federation – Multicluster von Haus aus

Einer der möglichen Wege zum verteilten und ausfallsicheren Betrieb eigener Anwendungen führt über kubefed – ein Kubernetes-Bordmittel. Mit dem Federation-Projekt beziehungsweise dem zugehörigen Kommandozeilentool kubefed lassen sich mehrere Kubernetes-Cluster an verschiedenen Standorten verbinden und so quasi als ein Cluster nutzen. Dennoch besteht weiterhin die Möglichkeit, die Cluster einzeln anzusprechen.

X-TRACT

- Mit Multi-Cloud-Szenarien lässt sich das Ausfallrisiko eines Datacenters oder Cloud-Anbieters weiter reduzieren.
- Kubernetes verfügt seit Längerem über eine Federation-Komponente, mit der sich mehrere Cluster gemeinsam ansprechen lassen.
- Angesichts der derzeit noch sehr dynamischen Weiterentwicklung der zugehörigen Komponenten ist vor dem Realeinsatz umfangreiches Testen Pflicht.

Bis zur Version 1.8 war die Federation-Komponente ein integraler Bestandteil von Kubernetes, danach gliederte man sie aus. Heute führt die hierfür gegründete Multicluster Special Interest Group sie als eigenständiges Projekt.

Wie Abbildung 1 zeigt, ist der Aufbau einer Federation denkbar einfach: Als Erstes muss man eine Control Plane in einen schon existierenden Kubernetes-Cluster ausrollen, also kubefed selbst. Es übernimmt in der Funktion als Control Plane die gesamte Steuerung der Verteilung der Deployments innerhalb der Cluster. Gleichzeitig überwacht es auch den Status der Deployments über alle verbundenen Cluster.

Die Control Plane benötigt gleichzeitig einen Nameservice, zum Beispiel Googles Cloud DNS oder Amazons Route 53. Denkbar wäre auch eine CoreNDS-Installation auf den Kubernetes-Clustern. Die Contrl Plane braucht den Nameservice, um die DNS-Einträge für die einzeln Services anzulegen beziehungsweise zu aktualisieren.

kubefed steuert die Federation über seinen Namespace `federation-system`. Ist die Control Plane erst mal in Betrieb, kann man nun weitere Kubernetes-Cluster hinzufügen, im Beispiel dieses Artikels in Listing 1 sind das AWS (Amazon Web Services) beziehungsweise EKS (Amazon Elastic Container Service for Kubernetes)

und GCE (Google Compute Engine) respektive GKE (Google Kubernetes Engine). Hat das für beide Cluster funktioniert, kann man sie sich mit dem Aufruf `kubectl get clusters` anzeigen lassen:

```
NAME          STATUS   AGE
aws-cluster1  Ready    5m
gce-cluster2  Ready    8m
```

Alternativ lässt sich die Federation selbst über den Namespace `federation-system` prüfen (siehe Listing 2 und 3). Über diesen kann man auch die eigentliche Applikation von kubefed sehen, die für Federation beziehungsweise die Steuerung am Ende zuständig ist. Listing 4 zeigt das Kubernetes `StatefulSet` mit dem dazugehörigen Service und dem Pod, in dem die Anwendung zur Kontrolle der Cluster in der Federation läuft.

Applikationen lassen sich nun mit dem richtigen Kontext über alle Cluster ausrollen und stehen dann entsprechend in jedem Cluster zur Verfügung. Hierfür erstellt man zunächst einen Federated Namespace `simple-ns`, in dem anschließend das Ausrollen der eigentlichen Anwendung erfolgen soll. Listing 5 zeigt die zugehörige YAML-Konfiguration. Das Generieren des Dienstes erfolgt über den Aufruf

```
kubectl create -f simple-namespace.yaml ⤸
                namespace "simple-ns" created
```

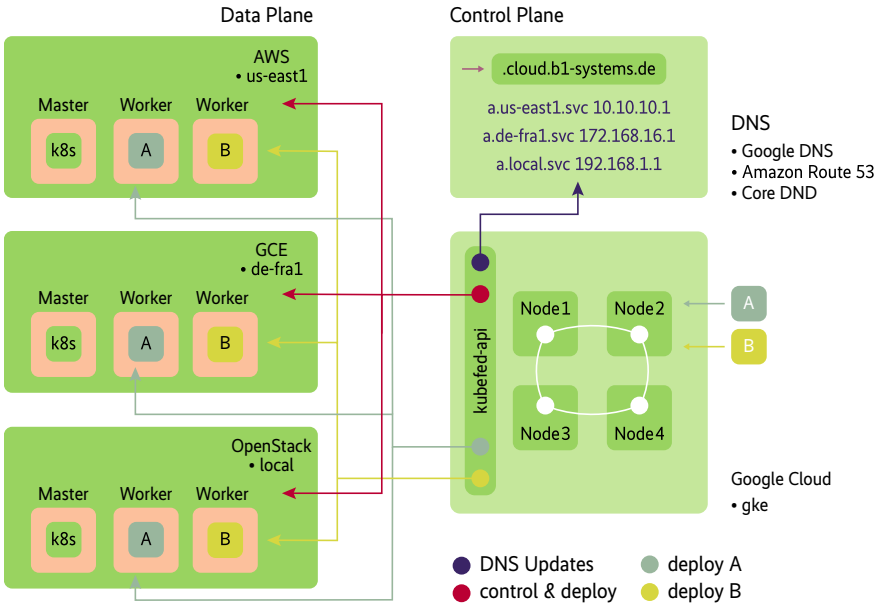
Anschließend lässt sich via kubectl kontrollieren, ob der Namespace auf beiden Clustern verfügbar ist (Listing 6). Wie man in der Bildschirmausgabe sieht, ist `simple-ns` sowohl auf dem GCE- als auch auf dem AWS-Cluster erstellt.

Als Nächstes soll ein einfacher NGINX-Webserver als auszurollende Beispielapplikation dienen. Hierfür muss man in der Datei placement.yaml zuerst das Placement, also die Verteilung der Applikation, festlegen (Listing 7).

Wie schon beim Namespace kommt hier eine spezielle Variante des Placements zum Einsatz, um die Verteilung auf beide Cluster zu beschreiben. Diese `Federated*`-Typen sind speziell für die Nutzung durch Kubernetes Federation gedacht. Hier besteht auch einer der Hauptunterschiede zu Federation V1. Bei V2 hat man sich dafür entschieden, eigene Typen einzuführen (Customer Resource Definition – CRD), statt die in Kubernetes vorhandenen Typen multiclusterfähig zu machen. Die daraus entstandene Struktur passt besser in das Konzept, wie sich Erweiterungen in die K8s-API einfügen.

Das eigentliche Ausrollen des Dienstes erfolgt über den oben angelegten Namespace `simple-ns`, die erforderliche Konfiguration findet sich in der Datei service.yaml (Listing 8). Auch hier lässt sich der Erfolg der Aktion per kubectl überprüfen (Listing 9).

Es werden nun natürlich auch Ingress-Controller gebraucht, damit der Inbound Traffic die Applikation erreichen kann. Im Beispiel service.yaml dient dazu `NodePort` – damit lauscht jeder Service auf einem lokalen Port im Cluster. Besser wäre es allerdings, `ClusterIP` oder `LoadBalancer` zu verwenden und das Ganze mit einem externen Load Balancer wie dem Google Cloud Load Balancing Service über alle Cluster zu verteilen.



Kubernetes Federation implementiert eine Kontrollschicht, die über die kubefed-api die Worker-Nodes der einzelnen Cluster steuern kann (Abb. 1).

Listing 1: AWS- und GCE-Cluster in die Kubernetes Federation aufnehmen

```
$ kubefed2 join aws-cluster1 --cluster-context aws-cluster1 --host-cluster-context ⤸
                                                    aws-cluster1 --add-to-registry --v=2
[...]
[...] Creating service account in joining cluster: aws-cluster1
[...]
[...] Created secret in host cluster named: aws-cluster1-229nv
[...] Created secret in host cluster: aws-cluster1
[...] Cluster credentials secret created
[...] Creating federated cluster resource
[...] Created federated cluster resource

$ kubefed2 join gce-cluster2 --cluster-context gce-cluster2 --host-cluster-context ⤸
                                                    aws-cluster1 --add-to-registry --v=2
[...]
[...] Creating service account in joining cluster: gce-cluster2
[...]
[...] Created secret in host cluster named: gce-cluster2-j4wc6
[...] Created secret in host cluster: aws-cluster1
[...] Cluster credentials secret created
[...] Creating federated cluster resource
[...] Created federated cluster resource
```

Projektausblick Kubernetes Federation

Die bislang innerhalb des Kubernetes-Projekts gepflegte Version Federation V1 wird nicht mehr weiterentwickelt. Die letzte Release stammt von Februar 2018. Es gibt aber mit der auf der KubeCon 2018 erstmals als Prototyp vorgestellten Federation V2 eine Nachfolgerin. Mittlerweile hat das Projekt Fahrt aufgenommen und wurde in Kubernetes Cluster Federation (KubeFed) umbenannt – ohne den Zusatz V2. Auch Red Hat hat diesen Teil des Kubernetes-Projekts aufgenommen und integriert kubefed seit der Version 4 in sein Produkt OpenShift.

Die im bisherigen Artikel behandelte Federation V2 von Kubernetes befindet sich in Entwicklung. Es ist in dieser Phase nicht zu empfehlen, sie produktiv einzusetzen. In Development- und Testlandschaften lohnt sich aber definitiv ein Blick.

Listing 2: Ausgabe von *kubectl get namespaces*

```
NAME                        STATUS   AGE
default                     Active   37h
federation-system           Active   84s
kube-multicluster-public    Active   84s
kube-public                 Active   37h
kube-system                 Active   37h
```

Listing 3: Ausgabe von *kubectl -n federation-system describe federatedclusters*

```
Name:         aws-cluster1
Namespace:    federation-system
Labels:       <none>
Annotations:  <none>
API Version:  core.federation.k8s.io/v1alpha1
Kind:         FederatedCluster
Metadata:
  Creation Timestamp:  2019-01-18T11:11:19Z
  Generation:          1
  Resource Version:    174067
  Self Link:           /apis/core.federation.k8s.i[...]
  UID:                 c80d747c-1b11-11e9-b5ce-0ae[...]
Spec:
  Cluster Ref:
    Name:  aws-cluster1
  Secret Ref:
    Name:  aws-cluster1-229nv
Status:
  Conditions:
    Last Probe Time:       2019-01-18T11:12:22Z
    Last Transition Time:  2019-01-18T11:11:42Z
    Message:               /healthz responded with ok
    Reason:                ClusterReady
    Status:                True
    Type:                  Ready
Events:                    <none>

Name:         gce-cluster2
Namespace:    federation-system
Labels:       <none>
Annotations:  <none>
API Version:  core.federation.k8s.io/v1alpha1
Kind:         FederatedCluster
Metadata:
  Creation Timestamp:  2019-01-18T11:12:26Z
  Generation:          1
  Resource Version:    174075
  Self Link:           /apis/core.federation.k8s.i[...]
  UID:                 f00b2076-1b11-11e9-b5ce-0ae[...]
Spec:
  Cluster Ref:
    Name:  gce-cluster2
  Secret Ref:
    Name:  gce-cluster2-j4wc6
Events:    <none>
```

Listing 4: Ausgabe von *kubectl -n federation-system get all*

```
NAME                                      READY   STATUS    RESTARTS   AGE
pod/federation-controller-manager-0       1/1     Running   0          99s

NAME                                      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/federation-controller-manager-service   ClusterIP   None         <none>        <none>    99s

NAME                                         READY   AGE
statefulset.apps/federation-controller-manager   1/1     99s
```

Für das erste Halbjahr 2019 sind Alpha- und Betaversionen geplant.

Alternativer Ansatz mit Istio und Spinnaker

Einen etwas anderen Ansatz verfolgt das im Folgenden beschriebene Verfahren mit Istio und dem Continuous-Deployment-Tool Spinnaker. Hier kommen zusätzlich gleich mehrere Komponenten zum Einsatz. Wie bei der oben vorgestellten Federation setzt diese Variante voraus, dass die zu verwaltenden Kubernetes-Cluster bereits aufgebaut sind. Istio ist ein Open-Source-Projekt, das einen Service Mesh Layer bereitstellt. Dieser soll das automatische Erkennen (Discovery) von Diensten innerhalb des Clusters erleichtern, damit nicht jede Anwendung bestimmte Netzwerkfunktionen einzeln umsetzen muss. Es arbeitet nicht nur mit Kubernetes, sondern auch mit HashiCorps Werkzeugen Nomad und Consul zusammen.

Istio lässt sich in einem einzelnen Cluster genauso einsetzen wie über Clustergrenzen hinweg. Dabei bildet es dann die Netzwerkkomponente, mit der man mehrere Cluster für Multi-Cloud-Umgebungen vernetzen kann. Aber Vorsicht: Istio ersetzt etwa nicht Flannel, das das Netzwerk innerhalb eines Clusters bereitstellt, sondern baut auf diesem auf.

Wie Abbildung 2 zeigt, besteht Istio aus mehreren Komponenten und läuft, wie schon erwähnt, auch in Kubernetes. Der große Vorteil ist, dass keine Modifikationen an den Anwendungen erforderlich sind, um Traffic-Flow, ACLs oder Metriken zu kontrollieren beziehungsweise bereitzustellen. Istio bietet ähnlich wie kubefed eher eine Control Plane für die Cluster; Spinnaker übernimmt das Deployment der Applikationen. Weitere Funktionen von Istio sind unter anderem:

- Dynamic Routing
- a/b-Tests
- Rate Limiting
- Health Checks

Für einen Einsatz installiert man auf dem Verwaltungscluster die Hauptkomponente, also den Master, während auf allen weiteren Clustern die Remotes, sozusagen die Slaves, ausgerollt werden. Diese bestehen aus verschiedenen Konfigurationen für den Cluster, der Komponente `Citadel`, wie sie auch auf der Control Plane existiert,

Listing 5: *simple-namespace.yaml*

```
apiVersion: primitives.federation.k8s.io/v1alpha1
kind: FederatedNamespacePlacement
metadata:
  name: simple-ns
  namespace: simple-ns
spec:
  clusterNames:
  - aws-cluster1
  - gce-cluster2
```

Listing 6: Ergebniskontrolle Federation

```
# Check aws-cluster1
kubectl --context=aws-cluster1 get ns
NAME             STATUS    AGE
ns/default       Active    3d
ns/kube-system   Active    3d
ns/simple-ns     Active    50s

# Check gce-cluster2
kubectl --context=gce-cluster2 get ns
NAME             STATUS    AGE
ns/default       Active    2d
ns/kube-system   Active    2d
ns/simple-ns     Active    1m
```

Listing 7: *placement.yaml*

```
apiVersion: primitives.federation.k8s.io/v1alpha1
kind: FederatedServicePlacement
metadata:
  name: test-service
  namespace: simple-ns
spec:
  clusterNames:
  - gce-cluster2
  - aws-cluster1
```

Listing 8: *service.yaml*

```
apiVersion: primitives.federation.k8s.io/v1alpha1
kind: FederatedService
metadata:
  name: test-service
  namespace: simple-ns
spec:
  template:
    spec:
      selector:
        app: nginx
      type: NodePort
      ports:
        - name: http
          port: 80
```

und dem sogenannten `sidecar-injector`. Die Remotes verknüpfen dann die Cluster miteinander. Dabei müssen in Version 1.5 von Istio die einzelnen Komponenten direkt miteinander kommunizieren können, Cluster unterschiedlicher Anbieter setzen hier also zwingend ein VPN voraus. Zukünftig soll es hier auch die Möglichkeit geben, über ein clusterinternes Gateway zu kommunizieren.

In jedem Pod wird ein Proxy als Sidecar-Container installiert, der den Traffic der Anwendungen durchschleift. Das eröffnet die Möglichkeit, ACLs und Metriken ohne Änderungen an der eigentlichen Anwendung umzusetzen. Die Steuerung des Sidecar-Containers erfolgt über den Istio-Dienst Pilot.

Für Metriken und Policies zeichnet der Dienst Mixer verantwortlich. Erstere können dann wieder in ein Prometheus laufen oder lassen sich über ein UI rudimentär abbilden. Weiter kann man mit OpenTrace Latenzen in Abläufen der Applikationen messen und visualisieren. Dabei stellt der Service Citadel die TLS-Zertifikate für die verschlüsselte Kommunikation über Cluster hinweg bereit.

Etabliert: Spinnaker und Istio

Darauf baut nun das Netflix-Projekt Spinnaker auf, mit dem sich Applikationen über Build-Pipelines gesteuert ausrollen lassen. Damit gehört es zu den Continuous-Deployment-Werkzeugen. Für den in diesem Artikel beschriebenen Fall ist es aber die perfekte Wahl, denn es kann im Rahmen einer Pipeline Applikationen in verschiedene Cluster ausrollen.

Bei Spinnaker heißt die Core-Applikation Orca. Sie verwaltet oder steuert Deployment, Builds, Metadaten und Notifications. Daneben existiert mit Gate eine umfangreiche API zum externen Steuern von Deployments. Auch das Spinnaker-WebUI Deck nutzt diese API.

Am Ende braucht aber Spinnaker auch noch den „Clouddriver“, damit es Kubernetes ansprechen kann. Neben Kubernetes gibt es noch diverse andere Endpoints, die man einsetzen kann: AWS, Azure, Google und OpenStack.

Zwar hat das kubefed-Projekt nur den Status alpha, aber im Kuberntes-Umfeld heißt alpha nicht unbedingt etwas Schlechtes. Es beschreibt lediglich den Stand der API etwa, dass sich einige Parameter noch ändern können. Im Kubernetes-Umfeld existieren viele als alpha deklarierte Schnittstellen die dennoch rege eingesetzt werden.

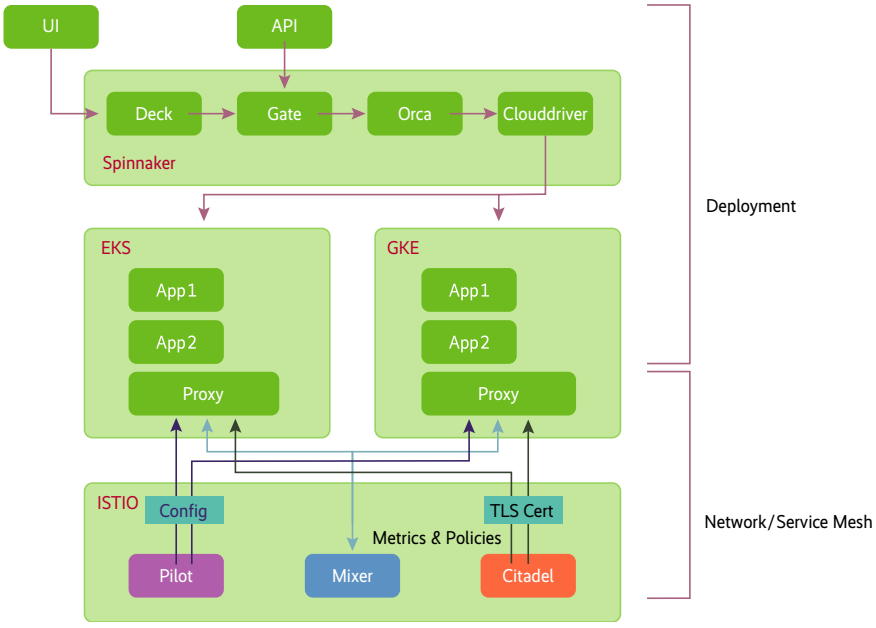
Listing 9: Erfolgskontrolle per *kubectl --context...*

```
kubectl --context=aws-cluster1 -n simple-ns get service
NAME           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
test-service   NodePort   10.107.32.199   <none>        80:30227/TCP   22s

kubectl --context=gce-cluster2 -n simple-ns get service
NAME           TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
test-service   NodePort   10.104.86.25   <none>        80:31941/TCP   23s

kubectl --context=aws-cluster1 -n simple-ns get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
test-deployment   3/3     3            3           24s

kubectl --context=gce-cluster2 -n simple-ns get deployment
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
test-deployment   5/5     5            5           24s
```



Istio und Spinnaker laufen nicht nur in Kubernetes, sie bieten auch einen der Federation ähnlichen Funktionsumfang (Abb. 2).

Wer sich nicht scheut, Deployments später noch mal zu ändern, kann durchaus schon auf kubefed zurückgreifen. Wer hingegen lieber auf Nummer sicher gehen möchte und kein Problem damit hat, mehr als eine Software aufzubauen, sollte zum Gespann Istio und Spinnaker greifen. Für alle Unentschlossenen bietet die Lernplattform Katacode einen 30minütigen Kurs zu kubefed (siehe ix.de/znz1).

Fazit

Auch wenn die Projekte auf einem guten Weg sind, ist es für potenzielle Nutzer derzeit noch schwierig, Multi-Cloud richtig umzusetzen. Die Verfügbarkeit der RZs bei den Cloud-Anbietern ist in 99,99 % der Einsatzfälle vollkommen ausreichend.

Das spiegelt sich auch in mangelndem Interesse an Kubernetes Federation wider, wie dessen Entwickler beklagen. Generell ist anzuraten, bei der Nutzung von Cloud-Diensten darauf zu achten, flexibel zu bleiben, um bei Bedarf schnell einen manuellen Failover anstoßen zu können.

(avr@ix.de)

Sebastian Meyer

arbeitet bei der B1 Systems GmbH als Linux Consultant und Trainer mit der Spezialisierung System-, Software- und Konfigurationsmanagement.

Michel Raabe

arbeitet als Consultant und Trainer mit den Schwerpunkten Cluster, Hochverfügbarkeit und Virtualisierung bei der B1 Systems GmbH.

André Steincke

arbeitet als Trainer und Consultant mit den Schwerpunkten XEN/KVM, Hochverfügbarkeit, Docker und Kubernetes bei der B1 Systems GmbH.