Unterstützt von

Cronon

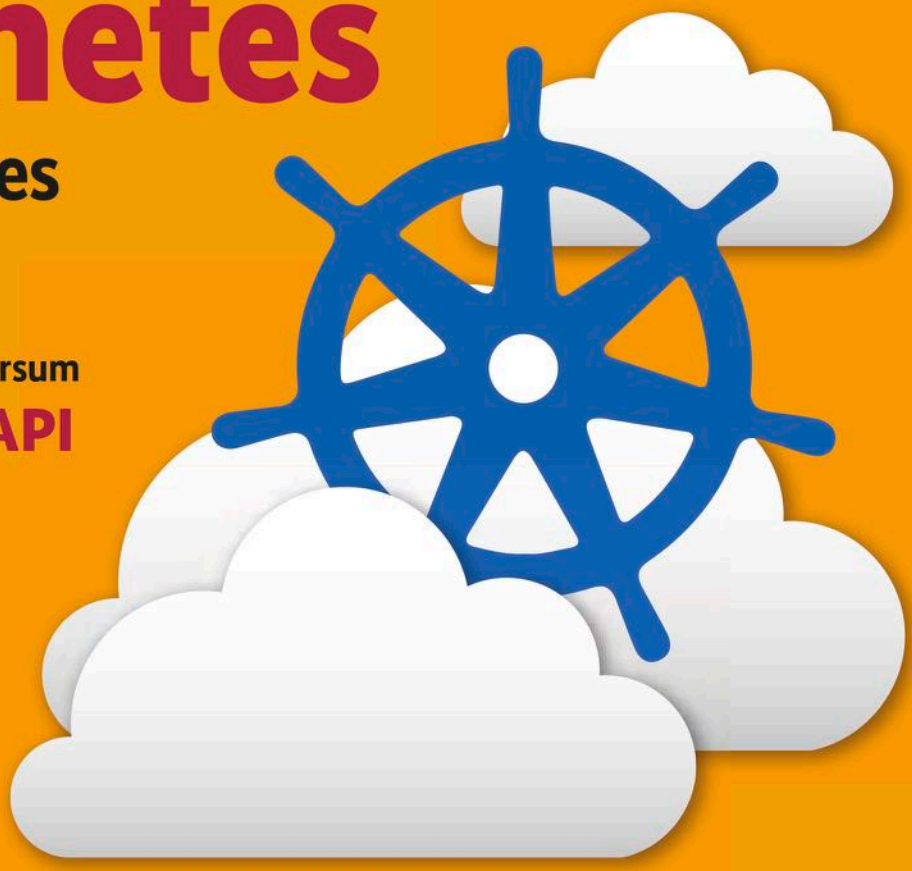
iX extra Juli 2022

Cloud

Kubernetes

Tools und Services

Eine Sonderveröffentlichung der Heise Medien GmbH & Co. KG



iX extra zum Nachschlagen:
www.ix.de/extra

CAPI: Rising Star im Kubernetes-Universum

Je mehr eine Kubernetes-Umgebung wächst, desto schwieriger und umfangreicher wird das Lifecyle-Management für die Cluster. Das Projekt Cluster-API verspricht Abhilfe.

In der IT ist Kubernetes als zentraler Container-Orchestrator unverzichtbar. Immer mehr Unternehmen setzen darauf und versprechen sich davon mehr Flexibilität für ihre Entwicklungsteams sowie bessere Skalierungsmöglichkeiten und damit verbunden eine deutlich verkürzte Time to Market. Das leuchtet sofort ein: Wenn Entwicklerinnen und Entwickler keine Zeit mehr darauf verwenden müssen, Infrastruktur anzufordern und aufzubauen, werden Kapazitäten frei, die in die eigentliche Entwicklung fließen können.

Ein sehr bewegter Markt

Dank der daraus entstandenen Dynamik ist der Markt der Kubernetes-Distributionen in den letzten Jahren regelrecht explodiert. Beispielsweise findet man auf GitHub bereits nach recht kurzer Recherche etliche wie Ranchers RKE2, k3s oder Kubermatics KubeOne sowie die bekannte Möglichkeit, mit kubeadm einen Kubernetes-Cluster zu installieren. Je nach Distribution bringt die Installation unterschiedliche Anforderungen und Komplexität mit sich. Bekanntlich bestehen Kubernetes-Cluster aus vielen komplexen Komponenten (kube-apiserver, kube-scheduler, kube-controller-manager, kubelet, etcd-Cluster und vieles mehr). Das aber erschwert deutlich den Einsatz klassischer Konfigurationsmanagementsoftware wie Ansible, Puppet und Salt – vor allem in Kombination mit den vielen unterschiedlichen Kubernetes-Distributionen in Sachen Lifecycle-Management.

Für jede Distribution mussten Developer bisher den Code selbst schreiben, um zumindest die Installation zu automatisieren. Im Internet finden sich zwar vorgefertigte Ansible Collections oder Terraform-Module, aber leider sind diese Code-Repositorys meistens veraltet oder es fehlen wichtige Features. Dadurch entsteht ein nicht unerheblicher Entwicklungsaufwand, zumal die Installation je nach Cloud-Plattform und deren SDK/APIs anders ausfällt und unterschiedliches Plattformwissen vorhanden sein muss oder aufzubauen ist. So kann es zum Beispiel passieren, dass den Terraform-Modulen oder Ansible Collections wichtige Features für die Provisionierung via VSphere oder andere Cloud-Plattformen fehlen. DevOps-Teams sind dadurch meistens gezwungen, eine Menge Code zu forken und zu pflegen, der für sie eigentlich nur Overhead bedeutet.

In Sachen Day-2-Operations stellen die vielen Distributionen den klassischen IT-Betrieb jedoch noch vor weitere Schwierigkeiten. Je mehr Cluster in einer Umgebung vorhanden sind, desto schwieriger und umfangreicher wird das Lifecycle-Management. Kritische Betriebssystem-Patches sowie regelmäßige Minor oder Major Updates für Kubernetes gehören hier weiterhin zum Alltag und erfordern eine zeitnahe, unterbrechungsfreie Provisionierung auf die Bestandsinfrastruktur,

Ausgewählte Kubernetes-Distributionen

Projekt	URL
Amazon EKS	aws.amazon.com/eks/eks-distro
Canonical Kubernetes	ubuntu.com/kubernetes
Google GKE	cloud.google.com/kubernetes-engine
k3s	k3s.io
KubeOne	github.com/kubermatic/kubeone
Microsoft AKS	azure.microsoft.com/de-de/services/kubernetes-service
Rancher RKE2	docs.rke2.io
Red Hat OpenShift	github.com/openshift
VMware Tanzu	tanzucommunityedition.io

CAPI-fähige Tools

Name	URL
clusterctl	cluster-api.sigs.k8s.io/clusterctl/overview.html
Crossplane	crossplane.io
EKS Anywhere	aws.amazon.com/eks/eks-anywhere/
Rancher 2.6	github.com/rancher/rancher/releases
Red Hat OpenShift (wird gerade eingebaut)	developers.redhat.com/products/openshift/overview
VMware Tanzu (Community Edition)	github.com/vmware-tanzu/community-edition

Listing 1: mgmnt-cluster.yaml

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraMounts:
    - hostPath: /var/run/docker.sock
      containerPath: /var/run/docker.sock
```

Listing 2: Start des kind-Clusters

```
[root@capi-demo capi]# kind create cluster --config mgmnt-cluster.yaml --name mgmnt
Creating cluster "mgmnt" ...
 ✓ Ensuring node image (kindest/node:v1.24.0)
 ✓ Preparing nodes
 ✓ Writing configuration
 ✓ Starting control-plane
 ✓ Installing CNI
 ✓ Installing StorageClass
Set kubectl context to "kind-mgmnt"
You can now use your cluster with:

    kubectl cluster-info --context kind-mgmnt

Have a nice day!
```

um sich vor Sicherheitslücken und Angriffen zu schützen.

Kubernetes goes Cluster-API

Abhilfe schaffen soll hier ein von der Kubernetes Special Interest Group (SIG) „Cluster Lifecycle" kuratiertes Subprojekt, das im Oktober 2021 die produktionsreife Version 1.0 der Cluster-API veröffentlicht hat. Die Cluster-API-SIG hat das Bereitstellen deklarativer APIs und Tools zur Vereinfachung der Provisionierung, des Upgrades und des Betriebs verteilter Kubernetes-Cluster zum Ziel.

Das Projekt behandelt die Konfiguration der Cluster sowie die Infrastrukturkomponenten wie Netzwerk, VMs und VPCs wie Application Workloads. Dies ermöglicht Multi-Cloud-Management von Kubernetes-Clustern und eine einheitliche Vereinfachung der Installation. DevOps-Teams steht somit eine einheitliche API zur Verfügung, die die generische Clusterinstallation vereinheitlicht. Sogenannte Provider vereinfachen und standardisieren das Installieren für verschiedene Cloud-Plattformen, das mit klassischen Konfigurationsmanagementsystemen erschwert wurde. Somit ist kein Spezialwissen über die Cloud-Provider-APIs aufzubauen und das unnötige Forking und die Wartung von Code entfällt. Die Lernkurve wird ebenfalls flach gehalten, da hier eine Kubernetes-Style-API zum Einsatz kommt. Sollte dennoch Bedarf bestehen, lässt sich das Framework jederzeit durch eigene Provider erweitern.

Bereitstellen der Cluster-API

Die Cluster-API erfordert zunächst einen funktionsfähigen Kubernetes-Cluster. Dieser

Listing 3: CAPI-Installation bei der Initialisierung

```
[root@capi-demo capi]# clusterctl init --infrastructure docker
Fetching providers
Installing cert-manager Version="v1.5.3"
Waiting for cert-manager to be available...
Installing Provider="cluster-api" Version="v1.1.3" TargetNamespace="capi-system"
Installing Provider="bootstrap-kubeadm" Version="v1.1.3" TargetNamespace="capi-↵
                                                    kubeadm-bootstrap-system"
Installing Provider="control-plane-kubeadm" Version="v1.1.3" TargetNamespace="capi-↵
                                                  kubeadm-control-plane-system"
Installing Provider="infrastructure-docker" Version="v1.1.3" TargetNamespace="capd-↵
                                                                  system"

Your management cluster has been initialized successfully!

You can now create your first workload cluster by running the following:

  clusterctl generate cluster [name] --kubernetes-version [version] | kubectl apply -f -
```

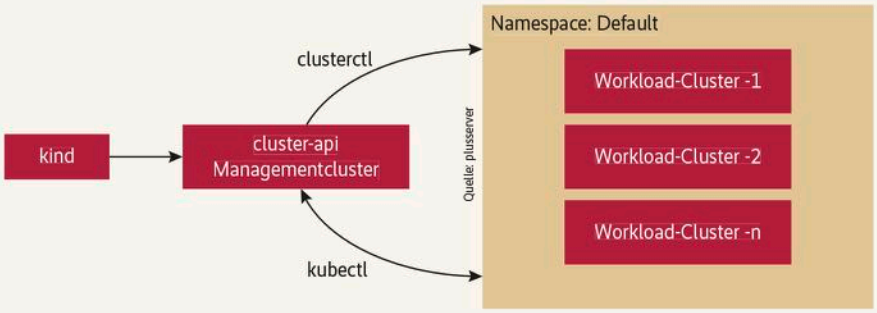
Listing 4: Ausgabe von kubectl get crd

```
NAME                                                              CREATED AT
certificaterequests.cert-manager.io                               2022-05-13T11:32:38Z
certificates.cert-manager.io                                      2022-05-13T11:32:38Z
challenges.acme.cert-manager.io                                   2022-05-13T11:32:38Z
clusterclasses.cluster.x-k8s.io                                   2022-05-13T11:32:56Z
clusterissuers.cert-manager.io                                    2022-05-13T11:32:38Z
clusterresourcesetbindings.addons.cluster.x-k8s.io                2022-05-13T11:32:56Z
clusterresourcesets.addons.cluster.x-k8s.io                       2022-05-13T11:32:56Z
clusters.cluster.x-k8s.io                                         2022-05-13T11:32:56Z
dockerclusters.infrastructure.cluster.x-k8s.io                    2022-05-13T11:33:00Z
dockerclustertemplates.infrastructure.cluster.x-k8s.io            2022-05-13T11:33:00Z
dockermachinepools.infrastructure.cluster.x-k8s.io                2022-05-13T11:33:00Z
dockermachines.infrastructure.cluster.x-k8s.io                    2022-05-13T11:33:01Z
dockermachinetemplates.infrastructure.cluster.x-k8s.io            2022-05-13T11:33:01Z
issuers.cert-manager.io                                           2022-05-13T11:32:38Z
kubeadmconfigs.bootstrap.cluster.x-k8s.io                         2022-05-13T11:32:57Z
kubeadmconfigtemplates.bootstrap.cluster.x-k8s.io                 2022-05-13T11:32:58Z
kubeadmcontrolplanes.controlplane.cluster.x-k8s.io                2022-05-13T11:32:59Z
kubeadmcontrolplanetemplates.controlplane.cluster.x-k8s.io        2022-05-13T11:32:59Z
machinedeployments.cluster.x-k8s.io                               2022-05-13T11:32:56Z
machinehealthchecks.cluster.x-k8s.io                              2022-05-13T11:32:56Z
machinepools.cluster.x-k8s.io                                     2022-05-13T11:32:56Z
machines.cluster.x-k8s.io                                         2022-05-13T11:32:56Z
machinesets.cluster.x-k8s.io                                      2022-05-13T11:32:56Z
orders.acme.cert-manager.io                                       2022-05-13T11:32:38Z
providers.clusterctl.cluster.x-k8s.io                             2022-05-13T11:32:32Z
```

zentrale Managementcluster hält die für den Lebenszyklus der Workload-Cluster benötigten Komponenten und Informationen vor. Hier laufen auch die zugehörigen Infrastruktur-Provider für die Plattformen AWS, Microsoft Azure, DigitalOcean, Docker, Hetzner, IBM Cloud, Google Cloud Platform, VMware vSphere sowie von OpenStack-Ressourcen, um dort Workload-Cluster zu erzeugen. Letztere sind nichts anderes als durch einen Managementcluster verwaltete Downstream-Cluster.

Die Cluster-API nutzt die Möglichkeit von Custom Resource Definitions (CRDs). Sie erweitern den Funktionsumfang der Kubernetes Control Plane, die durch ihre Komponenten die Kubernetes-API bereitstellt. Zu den wichtigsten CRDs gehören:

- **Machine** beschreibt die deklarative Spezifikation für eine Infrastrukturkomponente, die einen Kubernetes-Node hostet, etwa eine virtuelle Maschine. Sobald ein neues Maschinenobjekt erstellt wird, stellt ein providerspezifischer Controller einen neuen Host bereit und installiert ihn. Auf Basis der Maschinenspezifikation wird der neue Host als Knoten registriert. Wird die Maschinenspezifikation angepasst, so wird der Knoten neu erstellt. Der Node und die darunterliegende Infrastruktur lassen sich durch Löschen des Machine-Objekts entfernen.
- **MachineHealthCheck** definiert, ab wann ein Node als „unhealthy" gilt. Schlägt dieser Check fehl, wird der Node neu gebaut.
- **BootstrapData** enthält rollenspezifische Initialisierungsdaten (cloud-init) für Maschinen oder Knoten. Sie dienen zum Booten einer Machine in einen Knoten.



In der Beispielanwendung installiert kind (Kubernetes in Docker) zunächst einen lokalen Managementcluster und richtet die CAPI ein, um anschließend das automatisierte Deployment der Workload-Cluster anzuschieben (Abb. 1).

Um eine simple Bereitstellung als Demo testen zu können, werden folgende Server- und Softwarepakete vorausgesetzt: Eine VM mit Ubuntu 22.04 oder Alma Linux 8, kubectl 1.24.0, clusterctl 1.1.3, kind (Kubernetes in Docker) 0.13.0 und Docker:latest. Abbildung 1 zeigt einen Überblick.

Im ersten Schritt wird die benötigte Software installiert. In Schritt 2 wird der lokale Kubernetes-Cluster mit kind erstellt. Da kind für diese Demo Zugriff auf den Docker-Host benötigt, ist eine kind-Cluster-Definition anzulegen (siehe Listing 1).

Anschließend lässt sich der kind-Cluster starten. Listing 2 zeigt den erforderlichen Aufruf und die Bildschirmausgabe.

Der Managementcluster steht nunmehr bereit, es fehlt aber noch die Cluster-API. Sie lässt sich mit dem in Listing 3 gezeigten Aufruf installieren.

Anschließend sollte man zum einen mit dem Kommando `kubectl get ns` überprüfen, ob alle Komponenten vorhanden sind, und zum andern per `kubectl get pod -A` kontrollieren, ob sich tatsächlich alle Pods im Status Running befinden. Eine mit `kubectl get crd` abgerufene Übersicht über die installierten CRDs muss dabei wie in Listing 4 aussehen.

Damit steht die Cluster-API nunmehr zur Verfügung und es ist jetzt möglich, über die CRDs und clusterctl ein YAML-Manifest mit der Deklaration eines Kubernetes-Clusters zu generieren:

```
clusterctl generate cluster ↵
     cluster01 --flavor development ↵
               --infrastructure docker ↵
       --kubernetes-version v1.24.0 ↵
    --control-plane-machine-count=3 ↵
          --worker-machine-count=3 ↵
        > cluster01-clusterapi.yaml
```

Die generierte Datei cluster01-clusterapi.yaml (siehe ix.de/zqy9) enthält alle notwendigen Informationen zum Deployment des Clusters, das durch folgendes Kommando angestoßen wird:

```
kubectl apply ↵
        -f cluster01-clusterapi.yaml
```

Über den Befehl `kubectl get cluster` lässt sich der Erfolg der Aktion überprüfen.

Wer es lieber eleganter und ausführlicher hat, wählt den Weg über `clusterctl` (siehe Listing 5).

Als Nächstes ist ein Container Networking Interface (CNI) einzuspielen. Das erfordert eine valide kubeconfig-Datei für den Managementcluster, die sich per

```
clusterctl get kubeconfig ↵
     cluster01 > cluster01.kubeconfig
```

generieren lässt. Das Einspielen des CNI erfolgt dann einfach über einen Netzwerkprovider wie Calico mit

```
kubectl --kubeconfig=↵
       ./cluster01.kubeconfig apply ↵
  -f https://docs.projectcalico.org/↵
          v3.21/manifests/calico.yaml
```

Der Workload-Cluster ist damit installiert und lässt sich über `kubectl` beliebig skalieren. Auch andere Aufgaben wie Cluster-Upgrades kann man so erledigen. Diese Schritte lassen sich natürlich im Rahmen von CI/CD-Pipelines automatisieren, was ein sehr einfaches Multi-Cloud-Deployment von Workload-Clustern erlaubt.

CAPI-Einsatz in der Praxis

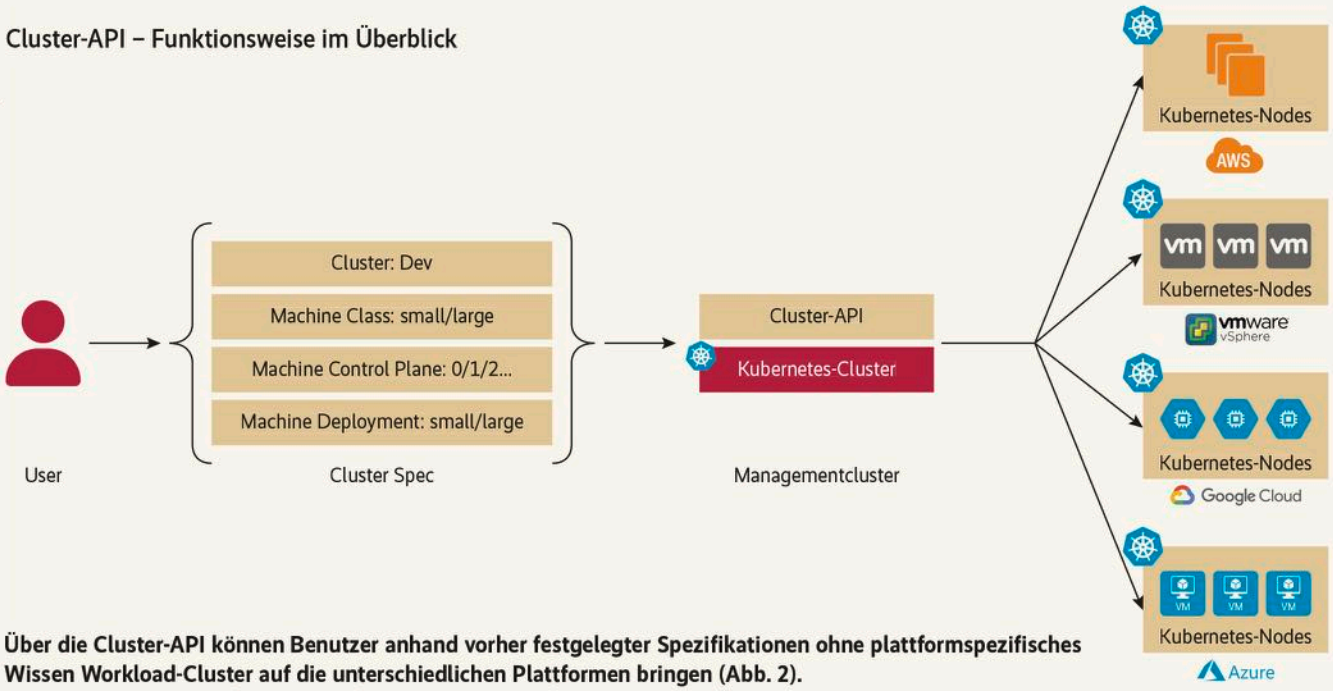
Neben der Möglichkeit, die Cluster-API in einem eigenen Kubernetes-Cluster zu betreiben, gibt es bereits Softwarehersteller und Open-Source-Projekte, die die Cluster-API per Default in ihre Infrastruktur integriert haben:

Anbieter und Kontributoren

Name	URL
Amazon	aws.amazon.com/eks/eks-distro
Apple	opensource.apple.com/projects/kubernetes/
AppsCode	appscode.com
D2iQ	d2iq.com
Daimler TSS	github.com/DaimlerTSS
Equinix	metal.equinix.com/developers/guides/kubernetes-cluster-api/
Ericsson	www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure/cloud-container-distribution
Giant Swarm	www.giantswarm.io
Google	cloud.google.com/kubernetes-engine
IBM	www.ibm.com/de-de/cloud/kubernetes-service
Intel	builders.intel.com/docs/networkbuilders/kubernetes-operators-automated-lifecycle-management-technology-guide.pdf [PDF]
Kubermatic	github.com/kubermatic/kubeone
Loft Labs	loft.sh
Mattermost	mattermost.com/solutions/use-cases/workflow-orchestration/
Microsoft	azure.microsoft.com/de-de/services/kubernetes-service
New Relic	newrelic.com/platform/kubernetes-pixie
Red Hat	www.redhat.com/en/topics/containers/what-is-the-kubernetes-API
Sovereign Cloud Stack	github.com/SovereignCloudStack/k8s-cluster-api-provider
Spectro Cloud	www.spectrocloud.com/blog/cluster-api-its-growing-up/
Talos Systems	www.talos.dev
Twilio	www.twilio.com/blog/tag/Kubernetes
VMware	tanzucommunityedition.io
Weaveworks	www.weave.works

Quelle: VMware

Cluster-API – Funktionsweise im Überblick



Über die Cluster-API können Benutzer anhand vorher festgelegter Spezifikationen ohne plattformspezifisches Wissen Workload-Cluster auf die unterschiedlichen Plattformen bringen (Abb. 2).

- **EKS Anyhwere** von AWS ermöglicht das Erstellen lokaler Developments und produktiver Kubernetes-Cluster auf VMware VSphere. Mit dem Kommandozeilentool eksctl erstellt man einen lokalen Bootstrap-Cluster (Managementcluster), der die Cluster-API bereitstellt. Darüber lassen sich die Workload-Cluster erzeugen.
- **Kubermatic** hatte bereits 2018 die Stärken der Cluster-API erkannt und in ihre gleichnamige Kubernetes-Cloud-Management-Plattform integriert. Das Unternehmen ist aktiver CAPI-Contributor. Die Entscheidung, die CAPI in die Plattform zu integrieren, basiert auf der Tatsache, dass APIs zuverlässig

sind und aus Entwicklungs- und Nutzersicht das Ausfallrisiko minimieren, da ein direkter Ausfall keinen Einfluss auf den verwalteten Workload-Cluster hat.

- **VMware** tritt ebenfalls als Mitwirkender in der Entwicklung der Cluster-API auf und integriert diese in den hauseigenen Tanzu Kubernetes Grid Cluster. Der zugehörige CAPI-Provider für den VMware Cloud Director CAPVCD stellt die Cluster-API an sich bereit sowie die Möglichkeit für Multitenancy. Dadurch können IT-Abteilungen isolierte Management- und Workload-Cluster bereitstellen. Zusätzlich werden multiple Control-Plane-basierte Managementcluster installiert, die High Availability für die Managementcluster forcieren und Ausfallsicherheit gewährleisten. Die Control Planes der Tanzu-Cluster nutzen dafür den hauseigenen NSX-T Advanced Load Balancer.
- Im Gaia-X-Kontext findet man die Cluster-API im bekannten **Sovereign Cloud Stack.** Der SCS stellt hierfür einen k8s-cluster-api-provider zur Verfügung (siehe ix.de/zqy9). Im Detail wird in diesem Projekt kind auf einer OpenStack-Instanz realisiert, die dann mit Terraform provisioniert wird. Der mit kind bereitgestellte Kubernetes-Cluster beinhaltet dabei die CAPI-CRDs und ermöglicht so das Erstellen von Clustern auf der SCS-Cloud.

Die Cluster-API ist erst der Anfang

Ein ebenfalls erwähnenswertes Open-Source-Projekt ist die Cloud-native Control-Plane-Plattform crossplane.io (siehe Abbildung 3). Es setzt den CAPI-Ansatz fort und erlaubt Provisionierung und Management von Cloud-Infrastrukturen jeglicher Art per Kubernetes-API. Mit Crossplane (wie auch mit der Cluster-API) lassen sich durch CRDs über die Kubernetes-API nicht nur die Kubernetes-Cluster selbst, sondern auch Datenbanken wie AWS RDS oder GCPs CloudSQL-Instanzen und mehr provisionieren.

Hierzu nutzt Crossplane sogenannte Provider, die für etliche Cloud-Plattformen, Software- und SaaS-Dienste wie AWS, GCP, GitLab, ArgoCD, Helm oder auch Terraform zur Verfügung stehen. Dadurch ermöglicht Crossplane operativen Plattformteams externe Infrastruktur-APIs zu konsumieren, ohne selbst irgendwelchen Code schreiben zu müssen. Das Open-Source-Projekt bietet mit Terrajet ein Code-Generator-Framework, um eigene Provider mittels Terraform zu designen.

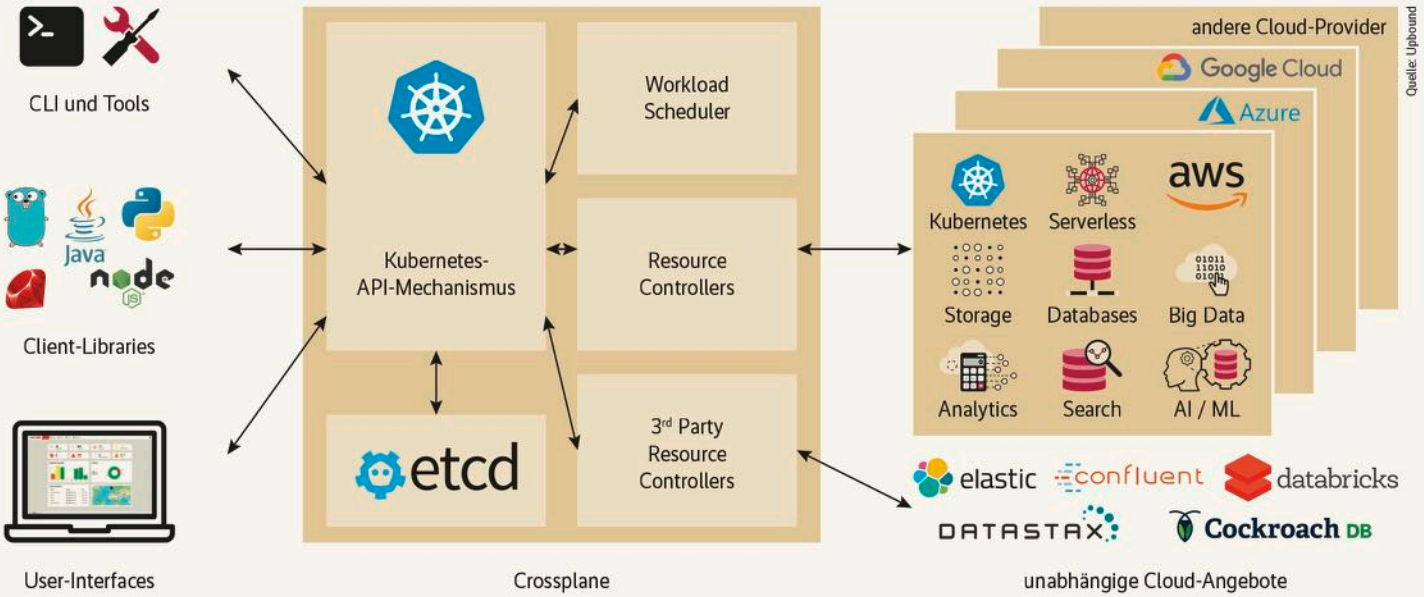
Potenzial für den skalierbaren Kubernetes-Betrieb

Die erweiterbare Kubernetes-API ist durch CRDs wie die Cluster-API und auch mit Crossplane auf dem Weg, die zentrale Multi-Cloud-native-Schnittstelle für Unternehmen zu werden. Die simple Installation und das vereinfachte, vereinheitlichte Lifecycle-Management für die Cluster sowie der strikte API-first-Ansatz spielen Unternehmen und hier besonders den DevOps-Teams in die Karten.

Grundsätzlich bedeutet dies aber auch, dass DevOps-Abteilungen sich verstärkt auf den Einsatz und Betrieb von Kubernetes spezialisieren müssen und bestehende, auf Ansible, Puppet und Co. basierenden CI/CD-Pipelines überdenken sollten. Der in den letzten Jahren entstandene deklarative GitOps-Ansatz ist eine mögliche praktische Lösung. Die Cluster-API lässt sich sehr gut mit Argo CI/CD oder GitLab CI/CD verwenden. Die Notwendigkeit des Einsatzes klassischer Systeme für das Konfigurationsmanagement bleibt zu einem Teil bestehen, da ja zumindest ein Bootstrap-Cluster erstellt werden muss. Ganz im Sinn des KISS-Prinzips (Keep it simple, stupid) kann die Cluster-API aber die Komplexität deutlich verringern und die gesamten Pipeline-Abläufe vereinfachen.

Listing 5: Ausgabe von clusterctl describe cluster cluster01

```
NAME                                                          READY  SEVERITY
Cluster/cluster01                                             False  Warning
├─ClusterInfrastructure - DockerCluster/cluster01             True
├─ControlPlane - KubeadmControlPlane/cluster01-control-plane  False  Warning
│ ├─2 Machines...                                             True
│ └─Machine/cluster01-control-plane-hfvh8                     False  Warning
└─Workers
  └─MachineDeployment/cluster01-md-0                          True
    └─3 Machines...                                           True
```



Mit Crossplane lassen sich nicht nur Kubernetes-Cluster, sondern auch andere Ressourcen provisionieren (Abb. 3).

```
REASON              SINCE  MESSAGE
ScalingUp           52m    Scaling up control plane to 3 replicas (actual 2)
                    52m
ScalingUp           52m    Scaling up control plane to 3 replicas (actual 2)
                    52m    See cluster01-control-plane-fcfb5, cluster01-control-plane-znbd6
BootstrapFailed     15m    1 of 2 completed

                    50m
                    52m    See cluster01-md-0-56dc9c854c-bt8n8, cluster01-md-0-56dc9c854c-g4pmp, …
```

Ganz zum Schluss sollte man ebenfalls nicht außer Acht lassen, dass ein erweitertes Securitykonzept sowie Backup-und-Recovery-Strategien für die Cluster-API zu entwickeln sind. Dafür gibt es von der SIG noch keine Best Practices. Diese Komplexität lässt sich aber ebenfalls einsparen, indem ein entsprechendes Produkt von VMware oder anderen Anbietern zum Einsatz kommt.

Fazit

Im Grunde geht es stets darum, Komplexität zu verringern, um Teams auf Entwicklungs- und Bereitstellungsseite zu entlasten. Container sowie deren Bereitstellung mit Kubernetes haben dazu erheblich beigetragen. Doch mit dem zunehmenden Bedarf (Stichwort Skalierung) wird auch Kubernetes zu einem Thema, das in Entwicklungsabteilungen unnötig viele Ressourcen bindet – speziell mit Blick auf die Bereitstellung. Noch immer ist spezifisches Wissen über SDKs der verschiedenen Hyperscaler erforderlich.

CAPI ist hier ein guter Ansatz, um Kubernetes „at Scale“ im Unternehmen betreiben zu können. Statt Hyperscaler-Experten zu bemühen, die entweder spärlich vorhanden sind oder sich anderweitig effizienter einsetzen lassen, bezieht man die Komponenten bequem über die universelle API. Dafür ist lediglich – wie beschrieben – die Cluster-API zu installieren. Alternativ bieten einige Plattformen wie VMware vSphere, Gardener oder OpenShift die CAPI bereits als integralen Bestandteil an – und fungieren zusätzlich als Control Plane, was das Multi-Cloud-Management erleichtert. (avr@ix.de)

Quellen

Download der Beispieldateien und weitere Links: ix.de/zqy9

Michael Vogeler
verantwortet seit 2019 als Head of DevOps bei plusserver die Koordination des zeitgerechten Aufbaus sowie der Inbetriebnahme von Systemumgebungen.

Kubernetes ist nicht gleich Multi-Cloud

Der Einsatz von Kubernetes gilt oft als Synonym für Multi-Cloud, garantiert aber nicht automatisch die Portabilität von Anwendungen. Für eine erfolgreiche Multi-Cloud-Strategie gilt es Randbedingungen zu erkennen.

Eines der heißesten Themen im Cloud- und DevOps-Umfeld ist zurzeit Multi-Cloud. Doch auch wenn laut Gartner die meisten Unternehmen nach wie vor einen Hyperscaler für ihre Cloud-Workloads präferieren, verteilen gut drei Viertel diese mittlerweile auf zwei oder mehrere Hyperscaler (76 %). Viele Unternehmen sind der Überzeugung, containerisierte Ansätze und der Einsatz von Kubernetes wären der Heilige Gral, um einen Lock-in bei einzelnen Hyperscalern zu vermeiden und ihre Applikationen zwischen zwei Clouds zu verschieben. Den Zahlen von Gartner zufolge funktioniert das in der Realität aber in weniger als zwanzig Prozent der Fälle.

Dieser Artikel zeigt, warum Kubernetes nicht automatisch die Portabilität einer Applikation zwischen unterschiedlichen Clouds garantiert, und geht darauf ein, welche Argumente wann für und wann gegen ein Multi-Cloud-Szenario sprechen und wie sich echte Portabilität und weiterer Nutzen mit Multi-Cloud erzielen lässt.

Wie kommt man zur Multi-Cloud?

IT-Verantwortliche kommen zum Multi-Cloud-Szenario oft wie die Jungfrau zum Kind: sei es durch Developer in Digitalisierungsprojekten, die einfach die gängigsten Cloud-APIs einsetzen, oder durch Zukäufe. So entsteht „Multi-Cloud by Chance", also unbeabsichtigt. Parallel dazu gibt es „Multi-Cloud by Strategy": den gezielten Einsatz unterschiedlicher Clouds, um Ausfälle zu vermeiden, unterschiedliche Funktionalitäten oder Kostenprofile zu nutzen oder Compliance-Vorgaben zu erfüllen.

Heutzutage, wo drei Viertel der Unternehmen mehrere Cloud-Provider für dieselbe IT-Lösungskategorie einsetzen, ist es oft erschreckend, wie wenig sich IT-Infrastrukturverantwortliche, Softwarearchitekten oder -entwicklerinnen mit Kosten-Nutzen-Abwägungen von Multi-Cloud-Umgebungen befassen. Trotz immer ausgereifterer Tools von Microservice-Meshes über API-Management bis hin zu Cloud-Brokerage-Plattformen: Die schiere Komplexität von Multi-Cloud-Szenarien in großen IT-Landschaften ist überwältigend und nimmt stetig zu.

Der beste Ratschlag ist, Multi-Cloud für gleiche Workloads möglichst zu vermeiden und sich auf eine beschränkte, kuratierte Anzahl an Cloud-Providern, -Services und -APIs zu konzentrieren. Das Ziel: digitale Lösungen so zu designen, dass sie eine oder möglichst wenige Clouds einsetzen, die „good enough" funktionieren und möglichst leicht zu managen sind.

Doch was ist, wenn das keine realistische Option ist, etwa weil das Unternehmen zu groß ist und zu viele unterschiedliche digitale Initiativen mit einer zu großen Vielfalt und Bandbreite bestehen? Dann bleibt nur, den Istzustand bestmöglich zu verwalten. Dabei wird Kubernetes häufig als einfache Lösung präsentiert. In der Praxis gibt es aber einiges zu bedenken.

Was bisher geschah …

Wir schreiben das Jahr 2008, erste Artikel über das Phänomen Cloud erscheinen in der deutschsprachigen Fachpresse. Begriffe wie Public und Private Cloud sind noch alles andere als etabliert, geschweige denn Hybrid- oder Multi-Cloud. AWS ist gerade sechs geworden und S3-Storage steht noch am Anfang seines zukünftigen Siegeszugs. Anfangs mit großer Skepsis betrachtet, setzen in den folgenden Jahren immer mehr Unternehmen auf die Innovationskraft der Cloud.

Im Zuge des AWS-Erfolgs treten weitere Anbieter auf den Plan, etwa Google, das Anfang der 2010er-Jahre bereits Datenzentren mit mehreren Tausend Servern besitzt und sein Know-how über Cloud-Computing zum Geschäftsmodell macht. Entscheidende Innovationen veröffentlicht der Suchmaschinenriese allerdings immer wieder als Open-Source-Projekte, so auch 2014 Kubernetes. Aus dem Wettbewerb der später als Hyperscaler bekannten Cloud-Provider entstehen weitere wegweisende Technologien, wegen ihrer elementaren Bedeutung auch „Commodity Services" genannt. Häufig bleiben diese jedoch proprietär und somit nur bei bestimmten Anbietern verfügbar. Entwicklerteams beginnen deshalb mit den unterschiedlichen Cloud-Anbietern zu experimentieren, um verschiedene Features zu nutzen und Kosten zu optimieren.

Multi-Cloud-Herausforderungen

Bei Multi-Cloud-Szenarien mit Kubernetes-Infrastruktur gibt es viele Faktoren, die einer vollständigen Portabilität zwischen Cloud-Providern im Wege stehen können. Allgemein gilt: Je mehr eine Recheninstanz auf Dienste eines bestimmten Anbieters angewiesen ist, desto schlechter steht es um die Portabilität. Selbst wenn man alle anbieterspezifischen Funktionen vermeidet, kommt man nicht umhin, sich mit unterschiedlichen Provisionierungs- und Konfigurationsprozessen zu befassen.

Es gibt auch externe Dienste, die für den Betrieb eines Systems entscheidend sein können, wie eine gemanagte Datenbank, deren Funktionen nur mit erhöhtem Aufwand in einem Kubernetes-Cluster nachgeahmt werden können (etwa über einen Database Operator). Ein anderes Beispiel ist das Identitätsmanagement, das sich zwar über Services wie Dex (OpenID Connector) abstrahieren lässt, aber die Abhängigkeiten zu Diensten wie Azure AD nie vollständig auflöst.

Viele grundlegende Cloud-Services, die auf den ersten Blick vergleichbar und kompatibel erscheinen, können verborgene Inkompatibilitäten haben, etwa Unterschiede bei Performance, SLA oder Parametern. So lassen sich für persistenten Storage beispielsweise sowohl der Blockspeicherdienst Elastic Block Storage (EBS) bei AWS als auch sein Bruder GCE Persistent Disk bei GCP mithilfe der Storage Classes und des Container Storage Interface (CSI) in Kubernetes einbinden. Bei AWS können Nutzer jedoch die I/O-Performance IOPS pro Gigabyte festlegen, während das bei GCP nicht funktioniert. Das kann dazu führen, dass die notwendige Diskperformance für den Workload nicht zur Verfügung steht und die Applikation sich anders verhält oder sogar Fehler produziert.

Stateful-Systeme wie Datenbanken oder ein Data Warehouse erschweren die Portabilität zusätzlich. Abgesehen von den enormen Kosten, große Datenmengen zu transferieren, können kleine Unterschiede in der Performanz oder im Durchsatz entscheidend sein.

Ist ein Teil der Infrastruktur nicht containerbasiert und Teil des Kubernetes-Clusters, wird Reengineering für die Portierung nötig, was mit größeren Schwierigkeiten verbunden ist. Darauf sollten DevOps-Teams gefasst sein.

Für eine gelungene Multi-Cloud-Applikationsimplementierung gilt es nicht nur, diese Unterschiede zu identifizieren und ihre Auswirkungen auf die Applikation zu untersuchen, sondern sie im Zuge der Evolution der Anbieter kontinuierlich zu evaluieren. Wie das aus praktischer Perspektive aussieht, zeigt der nächste Abschnitt.

Gute Absichten, schlechte Aussichten

Die Verlockungen der Multi-Cloud sind zahlreich: Ausfälle umgehen, die Vorteile von Container- und Cloud-native-Technologien nutzen, Regionen anbinden, die vielleicht nur bei einem Hyperscaler verfügbar sind, oder gar der Aufbau eines Kubernetes-Clusters, der eine Control-Plane über mehrere Datenzentren hinweg schafft. Egal, welches Vorhaben den Anstoß für Multi-Cloud by Strategy gibt, ab irgendeinem Punkt gerät die Implementierung meist ins Stocken.

Das folgende Szenario ist sicherlich für zahlreiche Unternehmen so oder so ähnlich typisch: IT-Verantwortliche nehmen sich vor, eine Plattform für Applikationsentwickler zu schaffen, damit diese ihren Code in Container-Images verpacken und zusammen mit einer kurzen YAML-Beschreibung deployen können, ohne sich mit den Feinheiten einzelner Clouds beschäftigen zu müssen. Das SecOps-Team hingegen ist naturgemäß bestrebt, für jede Cloud entsprechende Sicherheitsmechanismen, Backups und Monitoring-Möglichkeiten zu schaffen.

Das notwendige Bindeglied stellen dann häufig Plattformteams dar, auch SRE (Site Reliability Engineering) genannt, ein Ansatz von Google, der Überlegungen aus der Softwareentwicklung auf Infrastruktur- und Betriebsprobleme anwendet. Diese Teams sollen die Anfragen der Entwicklerinnen und Vorgaben seitens der Operations- oder Securityteams miteinander harmonisieren. Was zunächst nach einer gelungenen Arbeitsteilung klingt, entpuppt sich schnell als neues Problem. Die Zusammensetzung von Teams und die Zuständigkeiten können sich mit der Zeit ändern, was zwangsläufig zu Reibungsverlusten führt. Nicht zu vergessen die Perspektive des Managements, das Ansprüche im Hinblick auf die Kostenstruktur des Cloud-Betriebs oder des Rollouts neuer Funktionen stellt.

Alle diese unterschiedlichen Anforderungen lassen sich in der Regel nicht so leicht in Übereinstimmung bringen, da meist verschiedene Clouds betroffen sind. Von einheitlichen Schnittstellen bis hin zu Kompatibilitätsfragen gilt es viele Herausforderungen zu lösen, bis aus diesem Gewirr ein beherrschbares Ganzes wird, das den Namen Multi-Cloud verdient hätte.

Abgeleitet von den geschilderten Herausforderungen entstanden drei Guidelines, um Kubernetes als echten Multi-Cloud-Enabler zu nutzen (siehe Kasten „Wegweiser in eine bessere Multi-Cloud-Zukunft"). Diese sind architektur- und designagnostisch.

Diese Guidelines dürften auf der Multi-Cloud-Journey helfen, den richtigen Weg einzuschlagen. Gleichzeitig erhöhen sie jedoch auch die Komplexität einer Applikation. Ohne kontinuierliches und methodisches Testen wird die erfolgreiche Umsetzung von Multi-Cloud-Rekonfigurationen sonst nicht gelingen.

Praxisbeispiel: Petra und Klaus in der Multi-Cloud

Ein kleines praxisnahes Codebeispiel soll diese Punkte verdeutlichen. Es nutzt einige wohlbekannte und verbreitete Tools und Bibliotheken. Interessierte können es per GitHub-Repository nachstellen (siehe ix.de/zrrb). Terraform dient zur Provisionierung von Infrastrukturelementen über eine deklarative Beschreibung der Ressourcen, die mit vielen

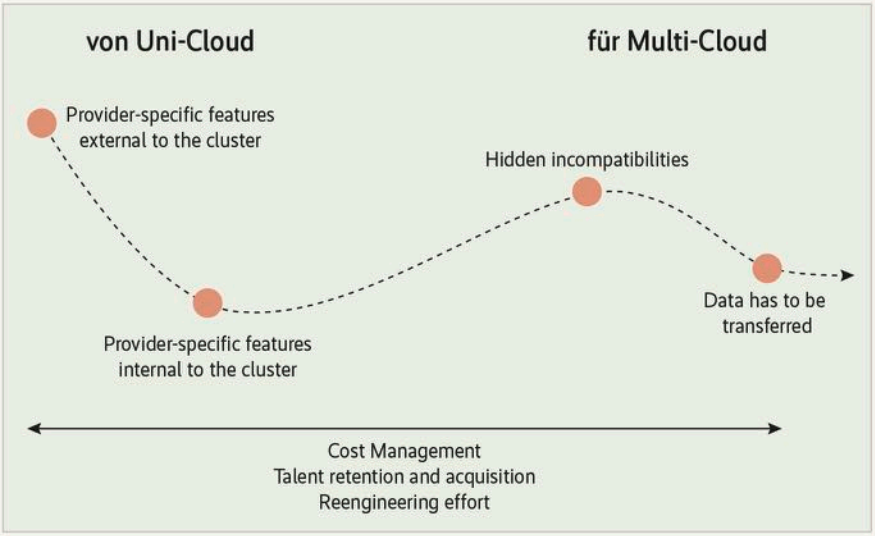
Kubernetes: Ein Haus mit festem Fundament?

Das Dilemma der Multi-Cloud sei hier mit einer kleinen Erzählung verdeutlicht: der Geschichte von den drei kleinen Schweinchen (Cloud-Architekten).

Der erste Cloud-Architekt baute seine Mono-Cloud aus Stroh, doch der böse große DDoS-Angreifer pustete sie einfach weg. Der zweite wusste es besser und baute seine Hybrid-Cloud aus Holz, in der Überzeugung, dass dieses Set-up standhafter sei. Doch auch hier dauerte es nicht lang und der große, böse und hungrige Workload brachte das Haus zum Einsturz. Beide Cloud-Architekten flüchteten sich schlussendlich in die Multi-Cloud zu ihrer Kollegin, die ihre Plattform auf ein Kubernetes-Fundament gebaut hatte.

Sie lachten erleichtert über ihr Erlebnis und wiegten sich in Sicherheit, doch die nächste unangenehme Überraschung in Gestalt des erbosten CFO ließ nicht lange auf sich warten. Als dieser, fuchsteufelswild über die ausufernden Kosten des Spaßes, durchs Fenster hereinschaute, bangten sie um ihre Karriere. Von da an beschlossen sie, guten Mutes gemeinsam an einer nachhaltigen und sicheren Lösung für ihr Cloud-Haus zu arbeiten.

Die Moral von der Geschichte: Viele Unternehmen sind der Überzeugung, sie seien mit Kubernetes und einer Multi-Cloud-Umgebung gut aufgestellt und könnten – da sie ja Cloud-native sind – ganz einfach zwischen Services und Providern hin- und herwechseln. Doch Kubernetes heißt nicht immer „Ende gut, alles gut".



Bei einem Wechsel von ausschließlich einem Provider hin zu einem Multi-Cloud-Ansatz kommen mehrere Faktoren ins Spiel (Abb. 1).

Wegweiser in eine bessere Multi-Cloud-Zukunft

Guideline 1:

- Nutzen und entwickeln Sie Module und Bibliotheken, die von vornherein mit mehreren Cloud-Providern kompatibel sind.
- Gibt es keine fertigen Module für anbieterunabhängiges Arbeiten, empfiehlt sich das Dependency-Inversion-Prinzip: Man baut ähnliche Module für alle Cloud-Anbieter und injiziert sie über ein einheitliches Interface in seine Lösung, etwa auf der Ebene Infrastructure as Code.
- Providerspezifische Informationen übergibt man dem Modul entweder zur Laufzeit als Parameter oder das Modul kann sie selbstständig aus der Umgebung ableiten, in der es gerade läuft.
- Es kann verlockend sein, für jeden Hyperscaler individuelle Infrastruktur-Stacks zu pflegen. Das führt jedoch zwangsläufig zu technischen Silos, die die Wiederverwendbarkeit der eigenen Lösungen reduzieren.
- Diese Guideline ist stark am Entwurfsmuster „Ports and Adapters" angelehnt und lässt sich analog implementieren. Es schlägt vor, dass Zugriffe auf externe Dienste über austauschbare Konnektoren erfolgen sollten, die die Anbieterlogik abstrahieren.

Guideline 2:

- Wo ein Dienst deployt wird, sollte für andere, darauf angewiesene Dienste keine Rolle spielen. Gleichzeitig sollte seine Schnittstelle immer gleich bleiben.
- Die Kommunikationslogik innerhalb des Ökosystems eines Anbieters sollte identisch mit der zwischen den verschiedenen Anbieter-Ökosystemen aufgebaut sein.
- Mit anderen Worten: Setzen Sie auf API-basierte Kommunikation zwischen unterschiedlichen Diensten.
- Fälle, in denen Zugriffsberechtigungen für Applikationen erteilt werden, die ausschließlich in einer spezifischen Ressource verfügbar sind (wie gemanagte Identitäten in Azure), sind zu vermeiden. Stattdessen sollte sich jeder Dienst explizit authentisieren, wenn er mit anderen Diensten kommuniziert.

Guideline 3:

- Die Funktionalität des Systems sollte gleich bleiben, unabhängig von der internen Organisation.
- Man sollte das System kontinuierlich im Hinblick auf Kriterien wie Verfügbarkeit, Latenz et cetera überwachen. Außerdem ist eine bewusste Entscheidung darüber zu fällen, wie eine ideale Organisation des Systems aussieht, ob man ausschließlich auf einen bestimmten Anbieter setzt oder ob es einen Anbietermix geben soll.
- Diese Entscheidungen sind im gesamten System durchzusetzen, indem Services zerstört und neu aufgebaut werden. Letzteres erfolgt gemäß dem bevorzugten Deployment-Muster, sei es Canary (inkrementelles Rollout), Blue-Green (partielles Testen neuer Releaseversionen mit verringertem Traffic beziehungsweise verringerter Nutzung) oder ein beliebiges anderes.
- Die Autoren bevorzugen den Begriff Reorganisation anstelle einer vollständigen Migration, da man sich jederzeit entscheiden kann, für eine bestimmte Funktion immer auf einen konkreten Anbieter zu setzen. Es kann beispielsweise sinnvoll sein, einen einzelnen Anbieter für das Identitäts- und Zugriffsmanagement zu nutzen.
- Der Begriff Entscheidung wird hier genutzt, um das Event zu beschreiben, das eine Reorganisation von Modulen triggert. Doch es wurde noch nicht näher festgelegt, wie und durch wen diese Entscheidung getroffen wird. Die Implementierungen können von einem vollständig automatisierten und dezentralisierten Prozess bis hin zu manuellen und zentralisierten Handgriffen reichen. Beim erstgenannten Ansatz managt jeder Teil des Systems sein eigenes Deployment bei dem am besten geeigneten Provider, während weiterhin die Kommunikationsfähigkeit mit anderen Services sichergestellt wird. Beim zweiten entscheidet ein SRE-Team über die bestmögliche Konfiguration des Systems und nutzt eine zentrale Registry, um den Standort jedes Dienstes zu speichern.

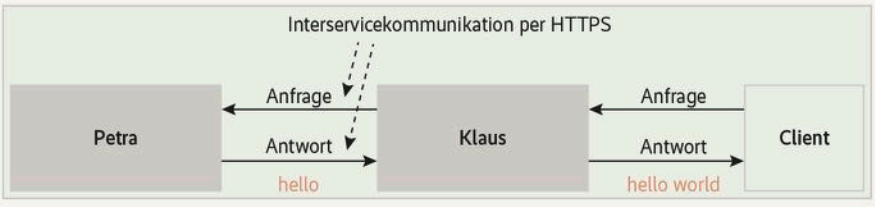
Cloud-Providern kompatibel ist. Die Bibliothek Terratest stellt die Muster und Helper-Funktionen für Infrastrukturtests bereit. Das Python-Webframework FastAPI dient im Beispiel als Mikroserviceapplikation. Das Deployment erfolgt auf den Managed-Kubernetes-Plattformen von Azure und AWS, Azure Kubernetes Service und Amazon Elastic Kubernetes Service.

Es gibt zwei Webservices, genannt Petra und Klaus. Petra besitzt nur einen Endpunkt, der stets `hello` ausgibt. Auch Klaus besitzt nur einen Endpunkt, der Petra aufruft und der Antwort das Wort `world` hinzufügt. Doch woher weiß Klaus, wo Petra sich befindet? Petras IP-Adresse ist als eine Umgebungsvariable von Klaus gesetzt. Dies kann beispielsweise über einen zentralen Konfigurationsmanager erfolgen.

Beide Webservices werden als Container in einem eigenen Kubernetes-Cluster deployt. Der Cluster ist im Terraform-Modul definiert, das so doppelt instanziiert wird. Das Modul erhält den String `provider` als Parameter, der entscheidet, ob der Cluster auf AWS oder Azure laufen soll. Für die Zwecke dieser Demonstration ist die Architektur des Systems so gestaltet, dass Klaus stets auf Azure läuft. Für diese Entscheidung kann es viele gute Gründe geben. In einer realen Multi-Cloud-Umgebung fällt diese Entscheidung situativ von Fall zu Fall.

Das gesamte System wird auf seine Multi-Cloud-Fähigkeit hin getestet. Es existieren zwei automatisierte Terraform-Tests: Der für die Module prüft, ob diese laufen, der für das System testet, ob der Webserver von Klaus `hello world` ausgibt. Das Unit-Testing für Klaus wird auf Azure, das für Petra sowohl auf Azure als auch auf AWS durchgeführt. Der Systemtest berücksichtigt zwei Konfigurationen – Konfiguration 1: beide Cluster laufen auf Azure; Konfiguration 2: Petras Cluster läuft auf AWS, der von Klaus auf Azure.

Auch wenn das Beispiel im Vergleich zu einer reellen Applikation extrem vereinfacht ist, dient es dazu, die praktische Bedeutung



Das Verhalten des Systems im Praxisbeispiel ist unabhängig von seiner internen Organisation konstant. Alle Dienste sind unabhängig und können bei unterschiedlichen Providern laufen (Abb. 2).

In iX extra 10/2022: Security: Neue Trends und Produkte zur it-sa

Immer perfidere Angriffsstrategien erfordern auch angepasste Gegenmaßnahmen. Hier hat sich das noch relativ junge Sicherheitskonzept Zero Trust als vielversprechend erwiesen. Es behandelt Zugriffe von außen wie von innen gleich, ohne internen Nutzern oder Geräten einen Vertrauensvorschuss zu gewähren. Die leider schlagkräftigsten Angriffe zielten jüngst auf die Supply Chain. Sie verursachten maximalen Schaden durch das Verteilen kompromittierter Software etwa als Updates. Ein geringer Aufwand für das Infizieren hatte katastrophale Auswirkungen. Hier sind Experten gefragt, die die Angriffe analysieren und Anleitungen zur Abwehr liefern. Einen anderen Schutz bieten Security Operations Center, die mit KI / maschinellem Lernen enorme Mengen an Netzverkehr auf Angriffe und Bedrohungen durchforsten.

All diese Ansätze sollen Unternehmen und Organisationen dabei unterstützen, aktuelle Herausforderungen und Risiken zu bewältigen. Orientierungshilfe geben auch die begleitenden Marktübersichten.

Erscheinungsdatum: 22.09.2022

Die weiteren iX extras

Ausgabe	Thema	Erscheinungsdatum
11/2022	Storage: Neue Storage-Techniken	20.10.2022
12/2022	Hosting: Colocation	24.11.2022

der vorgeschlagenen Guidelines zu erkunden. Die Konstruktion von Terraform-Modulen folgt aus der ersten Guideline, da das Modul den Provider abstrahiert, der für das Deployment der Ressource zum Einsatz kommt. Wie die Services miteinander kommunizieren, ist ein Beispiel für Guideline 2: Klaus interessiert es nicht, wo Petra läuft. Die Funktion des Systems, also die Ausgabe des altbekannten `hello world`, blieb gleich, wie in Guideline 3 vorgeschlagen. Die Entscheidung fiel in diesem Fall für einen manuellen Prozess, den die erneute Anwendung von Terraform umsetzt.

Jenseits von fortlaufendem und methodischem Testen sind sinnvolle Tooling- und Abstraktionslayer entscheidend, um die Komplexität zu reduzieren. Beispielhaft seien die folgenden Projekte genannt:

- **AI Sprint** stellt einen anbieterunabhängigen Abstraktionslayer für Edge-AI sowie ein Toolset für föderales Lernen und eingebettete Anonymisierung nebst anderen Anwendungen zur Verfügung.
- **Gaia-X:** Initiative für eine zukünftige Generation von Cloud-Infrastruktur, die Kompatibilität, Transparenz und europäische Datenhoheit verbessert.
- **Open Application Model:** portierbarer Standard für das Deployment von Applikationen mit klarem Fokus auf eine Auslieferung in hybriden Umgebungen.

Fazit

Bei Multi-Cloud-Deployments sollte man sich bewusst sein, dass allein Kubernetes keinesfalls die Portabilität einer Anwendung garantiert. Wenn dann nach wie vor mehr Gründe für die Multi-Cloud als dagegensprechen, sollten Systeme so konstruiert sein, dass sie echte Portabilität sicherstellen. Die Tools und Architekturmuster können eine Umgebung in ein komplexes Durcheinander verwandeln. Allein disziplinierte und methodische Arbeit wird langfristig zum Erfolg einer Multi-Cloud-Strategie führen. (avr@ix.de)

Quellen

Links zum GitHub-Repository und zu weiterführenden Informationen und Onlineartikeln: ix.de/zrrb

Leonardo Benitez
ist Data Scientist und Machine Learning Developer bei Skaylink.
Silvio Kleesattel
ist Technology und Innovation Lead bei Skaylink, einem führenden europäischen Cloud-Managed-Service-Provider.
Maximilian Zinke
ist Expert Consultant bei Skaylink im Bereich Kubernetes und Cloud-Native Solutions.