iX extra Juni 2020

Cloud

Container

Ein Kubernetes, sie zu binden ...

Orchestrierung im Wandel

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

Eine Sonderveröffentlichung der Heise Medien GmbH & Co. KG

Orchestrierung im Wandel

Ein Kubernetes, sie zu binden …

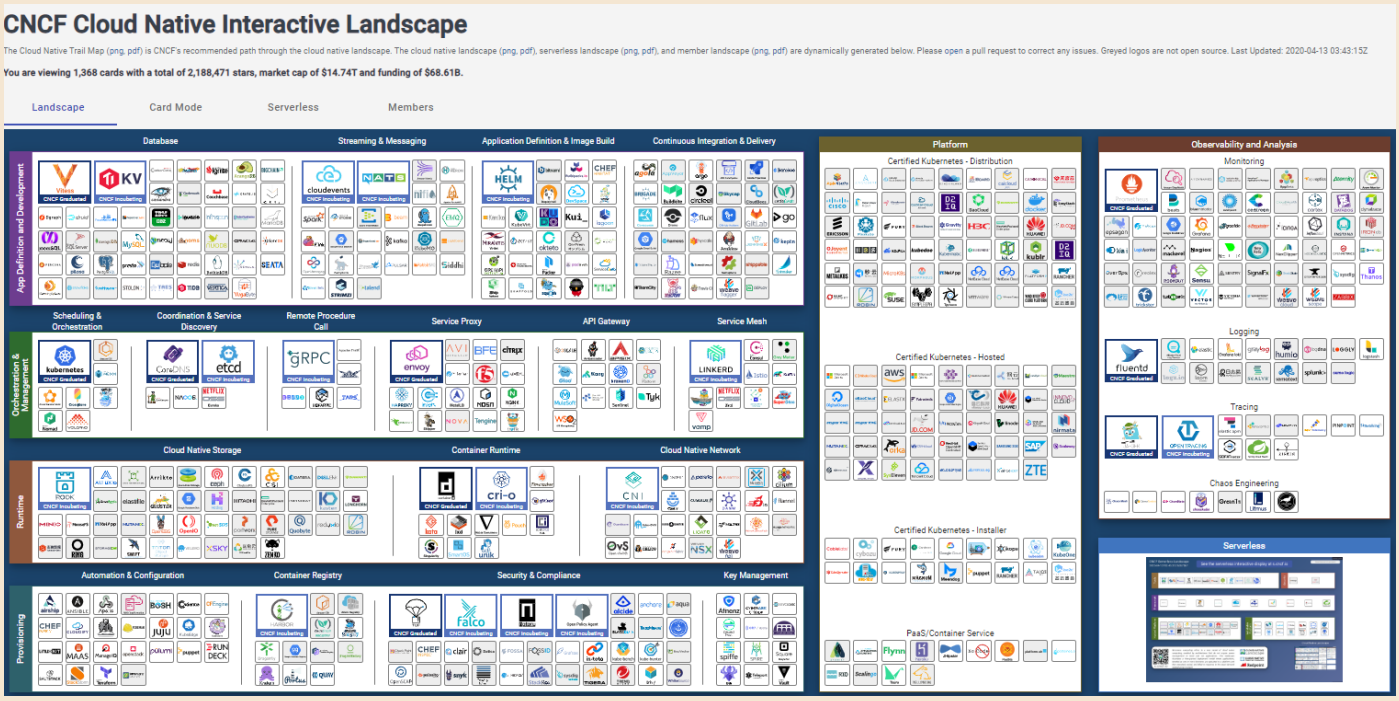
Vor sechs Jahren dachte man noch, Container wären die große Revolution. Dann kam Kubernetes und sorgte für die größte Zäsur in der IT seit dem Umstieg vom Host auf Windows-Server.

Die Ursprünge der Containertechnik liegen im letzten Jahrhundert. So wie Docker mit der Paketierung die Revolution auslöste, hat der tatsächliche Umbruch erst mit Kubernetes wirklich begonnen. Früher gab es noch viele Containerorchestrierungswerkzeuge. OpenShift, Rancher, VMware und Pivotal hatten – beziehungsweise haben teilweise immer noch – eigene Tools zum Überwachen und Steuern der Container auf den Containerhosts. Heute basieren faktisch alle auf Kubernetes (K8s) oder bieten es zumindest parallel mit an. OpenShift und Rancher haben ihre eigenen Entwicklungen vollständig aufgegeben, VMware hat Pivotal übernommen und mit deren Know-how und Expertise Kubernetes als Alternative zu vRealize und BOSH ins Portfolio aufgenommen. Ein Trend, dem auch SAP mit Gardener folgt, ist es, die Verwaltung der Kubernetes-Cluster auch wieder einem K8s-Master-Cluster anzuvertrauen. Das KoK (K8s over K8s) genannte Design erfreut sich auch bei großen Hostern wie Alibaba großer Beliebtheit.

Aber wie konnte das passieren? Warum waren nicht alle glücklich mit händisch von CD oder FTP installierten Servern, wo jeder Dienst in einem Start-und-Stopp-Skript in /etc/rc.d/ gesteuert wurde und man alle Änderungen an der Konfiguration mit dem Editor machte? Jeder, der dabei war, kennt die Antwort: Es war teuer, aufwendig, fehleranfällig, schlecht dokumentiert, schlecht zu warten und zu pflegen, und Patches einspielen war ein Glücksspiel mit Backup. Container, im Prinzip ja nur ein ursprünglich aus der Solaris-Welt stammendes „chroot on steroids“, boten damals schon eine kleine Verbesserung der Sicherheit. Auch Loopback-gemountete Laufwerke funktionierten bereits. Das Installieren einer Software wurde dadurch aber nicht einfacher, sondern komplexer. Gleiches gilt für die Überwachung und Steuerung, die sich mit den Diensten von systemd schon verbessert hatte, aber wiederum um den Preis gestiegener Komplexität. Für einen echten Fortschritt fehlte eine Paketierungslösung. Die gab es zwar eigentlich schon in jeder Distribution, aber RPM, DEB und Kollegen brauchten noch mehr oder weniger viele manuelle Eingriffe. Auch konnten sie vielleicht den Serverdienst installieren, aber die Konfiguration und die Inhalte, sei es Code, HTML, Bilder oder Datenbanken, waren in diesen Paketen nicht enthalten.

Mit den Dockerfiles auf Basis von Layered-Dateisystemen, eingebauter Überwachung und Steuerung, einem chroot für fast alle Namespaces des Kernels und einer einheitlichen Konfiguration brachte Docker Inc. hier die ersehnte große Erleichterung. Write once, run everywhere endlich auch für Server. Everything as Code, Unabhängigkeit von Hardware, Betriebssystem, anderen Diensten auf dem gleichen Server. Swarm, Dockers eigener Ansatz zur Steuerung, fand anfangs auch viele Anwender, aber wie auch bei eigenen Methoden anderer Hersteller erwies sich hier Kubernetes bald als die erfolgreichere Lösung.

Auch beim Kerngeschäft, dem Docker Daemon, haben andere Angebote die Führung übernommen. CRI-O, runc, containerd, podman und die anderen normalen Implementierun-



Quelle: landscape.cncf.io

Die Marktsituation im Cloud-Stack ist ein bisschen wie ein Dschungel aus Tools: Jeden Tag gibt es neue, alte verschwinden, und was sich genau durchsetzt, ist oft schwer vorherzusagen. Die Auswahl ist unübersichtlich, für vieles, bis auf die Orchestrierung selbst, gibt es Dutzende von Werkzeugen (Abb. 1).

gen des ehemals von Docker gesetzten Containerstandards sind heute marktbeherrschend. Dazu kommen auch die auf diesem Standard aufsetzenden Container-Runtimes mit einem Hypervisor-Typ-1-Ansatz wie FireCracker und Kata. Diese starten erst einen Kernel und dann den eigentlich Prozess im Container, sind also quasi eine VM in einem Container. Aber auch für die Runtimes gilt, wer nicht die Schnittstelle zu Kubernetes (CRI – Container Runtime Interface) implementiert, dürfte höchstens in einer Nische überleben. Dieser Artikel verwendet durchgängig die Bezeichnung Container, auch wenn Kubernetes diese in Pods zusammenfasst, da ein Pod immer mindestens einen Container enthält, aber auch mehrere enthalten kann. Das Konzept der Pods sieht dabei vor, dass alle Container in einem Pod sich alle Namespaces und das Netzwerkinterface teilen. So ist es einfacher, kleine Einheiten zu bauen, die wirklich nur eine Aufgabe übernehmen, sich aber über Speicher, IPC (Inter Process Communication) und localhost koordinieren. Die Pods sind einer der Gründe, warum Kubernetes den gesamten Markt aufgerollt hat, da sie es sehr erleichtern, austauschbare Mini-Dienste für alle Pods zu definieren, die Kubernetes dann teilweise nachträglich in die Pods einfügt. Solche Sidecars übernehmen dann beispielsweise die Authentifizierung und Verschlüsselung innerhalb eines Microservice oder Service Mesh.

Fressen oder gefressen werden

Im Dschungel der Tools rund um Cloud-native hilft die CNCF (Cloud Native Computing Foundation) mit Zertifizierungen und einer Übersicht – wenn man eine Liste von fast 1400 Einträgen Übersicht nennen möchte. Immerhin sind hier 14 Billionen USD Marktwert und 64 Milliarden Finanzierung aufgeführt. Diese immensen Summen verdeutlichen auch, dass ein Cloud-Stack kein neues Spielzeug, sondern die Zukunft der IT ist.

Man muss sich hier bewusst sein, dass Kubernetes kein fertiges Produkt ist, sondern eher ein Kessel Buntes. Es macht einiges selbst, aber manche Kernfunktionen gibt es nur als Plug-in-Interface. Diese Plug-ins muss der Administrator selbst auswählen und konfigurieren, etwa für den Speicher für die Anwendungen. Die wichtigste Komponente ist aber das Netzwerk, also das in das Kubernetes integrierte SDN, das dafür sorgt, dass die Container miteinander reden können.

Und selbst dieses Plug-in-Interface ist noch erweiterbar. Kubernetes arbeitet nach dem Prinzip der deskriptiven Konfiguration. Also sind keine Shellskripte im Einsatz, sondern die Konfiguration beschreibt einen Wunschzustand und Kubernetes übernimmt die Aufgabe, den Istzustand auf den Wunschzustand zu bringen. Die Konfiguration speichert Kubernetes in einer Key-Value-Datenbank, einem NoSQL-Speicher, der beliebige Schlüssel mit Werten füllen kann. Das standardmäßig verwendete etcd ist so tief in Kubernetes integriert, dass ein Austausch gegen ein anderes Tool nicht funktioniert. Was allerdings geht, ist, beliebige Konfigurationen über das K8s-Interface zu speichern. Diese Custom Ressource Definitions (CRD) erlauben es, Kubernetes beliebig zu erweitern. Sobald eine CRD hinterlegt ist, ruft Kubernetes bei einer Änderung eines Wertes in diesem CRD eine externe Software auf, die sich darum kümmert, dass der Wunschzustand erreicht oder eine Fehlermeldung ausgegeben wird.

Für einige Defaultaufgaben haben sich wenige Tools sehr gut durchgesetzt und gelten fast als Quasistandard. Die Auswahl ist groß, daher ist ein Standard auch nicht zu definieren, da jeder sich sein eigenes Kubernetes zusammenbasteln kann. Der Kasten „Verbreitete Kubernetes-Tools“ listet einige Werkzeuge, die sich bei den meisten Distributionen außer OpenShift finden. Allerdings ist das nach wie vor ein sehr volatiler Markt, und wer heute sagen kann, was in einem Jahr in dieser Liste auftauchen wird, sollte entsprechende Aktien kaufen.

Verbreitete Kubernetes-Tools

- Netzwerk: Calico mit Kiali als Web-UI
- Monitoring: Prometheus mit Grafana
- Protokollierung: ELK – Elasticsearch mit Kibana
- Code- und Artefaktverwaltung: Harbor oder GitLab
- Storage: entweder ein nativer Treiber des SAN-Anbieters oder GlusterFS oder Ceph
- Datenbanken: Cassandra, MongoDB
- Caching: Redis
- Mikrosegmentierung mit Service Mesh: ISTIO
- Sicherheitstools: Clair oder ein kommerzieller Anbieter

Hosting: Public, Hybrid, Private Cloud

Der Verdrängungswettbewerb ist dementsprechend hart. Nicht nur bei den Werkzeugen, auch die Anbieter von Hosting sind hier durch die Standardisierung noch stärker vergleichbar, das Vendor Lock-in hat abgenommen und Workloads ziehen schneller von einem Anbieter zum anderen. Und wenn das Internet eines bewiesen hat, dann dass es Monopole bevorzugt. Die drei Marktführer haben den Kuchen fast vollständig unter sich aufgeteilt, auch weil sie neben den CPUs Hunderte von Zusatzdiensten anbieten, die sich gut in die eigenen Anwendungen integrieren lassen. Wer hier schlau ist, mietet CPUs für seine Cluster preiswert beim kleinen Anbieter oder stellt sie in den eigenen Keller und nutzt die APIs der großen Anbieter

CNCF-zertifizierte Kubernetes-Serviceprovider

Firma	Homepage
Accenture	www.accenture.com/us-en/service-application-containers
Alibaba Cloud	us.alibabacloud.com
Camptocamp	www.camptocamp.com/en/solution/containers-orchestration/
Canonical	jaas.ai/canonical-kubernetes
Data Essential	www.data-essential.com
Datadrivers	www.datadrivers.de
Dell	www.dellemc.com/en-us/services/consulting-services/cloud-native-applications.htm
Deloitte	www2.deloitte.com/us/en/pages/technology/articles/cloud-container-technology.html
Giant Swarm	www.giantswarm.io
IBM	www.ibm.com/cloud/kubernetes-service/
inovex	www.inovex.de/de/leistungen/cloud/kubernetes/
Kinvolk	kinvolk.io
Loodse	www.loodse.com
Microsoft	azure.microsoft.com/en-us/services/kubernetes-service/
NetApp	nks.netapp.io
noris network	www.noris.de/it-services/cloud-services/paas/kubernetes/
PRODYNA	www.prodyna.com/en/cloud-native-computing-docker-kubernetes
Puzzle	www.puzzle.ch/de/home
Rackspace	www.rackspace.com
Rancher Labs	rancher.com
SAP	cloudplatform.sap.com/index.html
Storm Reply	reply.com/storm-reply/en/kubernetes
SUSE	www.suse.com
teutoStack	teutostack.de/produkte/managed-kubernetes/
VMware	cloud.vmware.com/vmware-enterprise-pks
VSHN	vshn.ch/en
WhizUs GmbH	www.whizus.com

über VPN. Nur bei einem Anbieter zu sein, ist entweder gesetzlich erzwungen oder fast schon uncool. Wenn der CTO beim Stammtisch mit den anderen CTOs nicht sagen kann, dass er auch Multi-Cloud macht, ist man nicht mehr in-crowd.

Worüber reden wir eigentlich, wenn wir „Cloud" sagen? Früher gab es den Begriff ASP – Application Service Provider. Das wäre heute SaaS, Software as a Service. Ein Anbieter betreibt die Server, das RZ und die Anwendung und der Kunde nutzt diese Software über eine Schnittstelle. Früher vt150 und heute Web. Früher war also nicht alles besser, aber meistens war es schneller. Damals stellte man Server bei Co-Location auf oder mietete sie, heute heißt das IaaS, Infrastructure as a Service. Früher gaben Unternehmen einem Dienstleister eine Software, damit der sich um den Betrieb kümmert, heute nennt sich das PaaS, Platform as a Service. Was ist also Cloud? Irgendwas mit Internet. Der Begriff hat eigentlich seine Bedeutung verloren. Wovon dieses *iX* extra redet, ist der Cloud-native-Stack. Also von Containern und ihrer Steuerung, über Software-defined Networking und die ganzen neuen Werkzeuge, die manche auch als Hipster-Hosting bezeichnen. Allen gemeinsam ist, dass sie Daten nur noch über HTTP austauschen, YAML und JSON als Formate nutzen, in der Standardinstallation keinerlei Sicherheit mitbringen und das Team dahinter agil arbeitet.

Nachdem das „Was" geklärt ist, wäre noch das „Wo" zu definieren. Wenn also „Cloud" irgendwas mit Internet ist, was ist dann eine „Private Cloud"? Computer, die man selbst gekauft hat und die im eigenen RZ und Netzwerk stehen. Warum ist das dann Cloud? Genau wegen Cloud-native – die Firma wendet die Technologien, die die großen Anbieter entwickelt

Kubernetes-Distributionen für On-Premises-Nutzung

	Google Anthos	VMware vSphere 7 / Tanzu	Docker EE	Red Hat OpenShift	SUSE Caas Platform	Rancher 2
Installation	○	○	⊕	○	⊕	⊕
IAM-Anbindung	⊕	○	○	⊕	○	○
Multi-Cloud	⊕	⊕	⊖	○	○	⊕
Logging	⊕	○	⊕	⊕	○	○
Monitoring	⊕	⊕	⊕	⊕	○	⊕
Storage-Integration	○	⊕	⊖	⊕	⊖	⊕
SDN	○	⊕	○	○	○	⊖
Serverless	⊕	⊖	○	⊕	⊖	⊕
Security	⊕	⊕	⊕	⊕	⊕	⊖
Upgrading	○	○	○	⊕	○	○
Scaling	⊕	⊕	○	○	○	⊕
Service Mesh	⊕	⊕	○	⊕	⊕	○

⊖: verbesserungsfähig; ○: funktioniert wie erwartet; ⊕: besonders positiv

Kommerzielle Cloud-Monitoring-Tools

Produkt	Webseite
Amazon CloudWatch	aws.amazon.com/de/cloudwatch/
AppDynamics APM	www.appdynamics.com/product/
Azure Monitor	azure.microsoft.com/de-de/services/Monitor/
Bitnami Stacksmith	bitnami.com/stacksmith
BMC TrueSight	www.bmcsoftware.de/it-solutions/truesight.html
CA UIM	www.ca.com/de/products/ca-unified-infrastructure-management.html
CloudEye (CES)	open-telekom-cloud.com/de/produkte-services/cloud-eye
CloudMonix	cloudmonix.com
Datadog	www.datadoghq.com/pricing/
Dynatrace	www.dynatrace.com/platform/cloud-infrastructure-monitoring/
LogicMonitor	www.logicmonitor.com
Monitis	www.monitis.com/de
Microsoft OMS	azure.microsoft.com/de-de/resources/videos/operations-management-suite-oms-overview/
New Relic (APM)	newrelic.com
Stackify Retrace	stackify.com/retrace/
VMware vRealize Hyperic	www.vmware.com/de/products/vrealize-hyperic.html

haben, selbst auf den eigenen Rechnern an. Da liegt es nahe, die Welten zusammenwachsen zu lassen. Multi-Cloud ist daher der Megatrend, der aber aktuell viele überfordert, die bereits mit einem Kubernetes-Cluster vollauf beschäftigt sind. Der erste Cluster steht dabei entweder bei einem der großen Public-Cloud-Anbieter oder in der eigenen Private Cloud.

Ein Sonderfall für den Betrieb von Cloud-native ist die Air-Gap-Installation. Die meisten Cloud-native-Werkzeuge brauchen Internet wie ein Fisch das Wasser. Ständig wird dynamisch nachgeladen, Bibliotheken, Container- und Betriebssystem-Images, manchmal sogar Konfigurationen und ganz oft die Installationsroutine. Viele Installationsanleitungen beginnen mit

```
root@server# wget http://some.
     io/install.sh | /bin/bash
```

Also Herunterladen eines Skripts von einer Webseite und sofortiges Ausführen als root. Das ist sicherheitsbewussten Anwendern natürlich ein Dorn im Auge und in vielen Umgebungen nicht tragbar. Hier zeigt sich oft die Schwäche des Ansatzes, eine im und für das Internet entwickelte Technologie in einem abgeschlossenen Netzwerk ohne Verbindung mit der Außenwelt zu betreiben. Es muss dafür ein kleines Internet geschaffen werden. Nicht nur DNS, sondern auch alle Speicher für die besagten Bibliotheken und Images sowie die vielen kleinen Hilfsdienste, die im Internet verfügbar sind, müssen vorhanden sein. Also ein nicht immer einfaches Unterfangen und um Größenordnungen schwieriger, als einen Server von CD zu installieren.

Komplexität

In der Praxis haben viele Unternehmen versucht, die eigenen Mitarbeiter mit der neuen Technik zu befreunden, und nach einiger Zeit des Experimentierens rufen sie Berater dazu, um dem scheiternden Projekt auf die Sprünge zu helfen. Die gestandenen Administratoren mit viel Berufserfahrung sind oft von dieser neuen Welt überwältigt, wo das über Jahre und teilweise Jahrzehnte gesammelte Fachwissen plötzlich nur noch wenig wert ist. Sein Linux bis zum Syscall gut zu kennen, ist toll, aber Systemadministratoren, die schon systemd verflucht haben, weil er ihnen zu komplex war, werden sich mit Kubernetes wahrscheinlich nicht anfreunden können. Die neue Welt ist sehr verzahnt, kompliziert, mehrfach überlagert und die Fehlersuche überfordert oft auch die Profis.

Schon die Beantwortung der einfachen Frage „Wo ist das Netzwerkpaket geblieben?“, die Anwendungsentwickler oft an die Firewall-Admins stellen, ist ein Unterschied zu früher. Damals, in der „guten alten Zeit“, gab es Kabel vom Server zum Switch, von da zur Firewall und dann weiter in Kabeln zu anderen Servern. Dann kam VLAN und es war nur noch ein Kabel, in dem viele virtuelle Netzwerke liefen. Aber die Frage, wo das Paket geblieben ist, ließ sich relativ einfach beantworten. Für den Anwendungsentwickler war es das Netzwerk oder die Firewall, für die Netz-Admins natürlich die Anwendung, die den Fehler verursacht hat. Aber nach einer Zeit des gemeinsamen Suchens wurde der Fehler meistens gefunden. Im Cloud-native ist das Kabel zwar noch da, aber darin wird ein Ethernet-Frame transportiert, in dem ein VXLAN-Paket steckt, in dem ein IP-Paket steckt, in dem ein IP-Paket steckt. Und das betrachtet den physischen und den Anwendungslayer noch nicht einmal. Beschrieben ist hier ein typischer Aufbau. Also die Kabel und das Ethernet sind die Basis. Darüber liegt ein SDN mit VXLAN, das die virtuellen Server miteinander verbindet, auf denen die Container laufen. Diese reden miteinander über ein weiteres SDN, das bei Kubernetes ein Plug-in gemäß CNI-Standard (Container Networking Interface) ist. Hier im Beispiel ein Calico mit IP over IP, da man keinen VXLAN-Frame in einen anderen VXLAN-Frame stecken kann. Andere Beispiele sind auch möglich, die Auswahl ist groß.

Also zur ursprünglichen Frage zurück: Wo ist das Paket? Tja, es könnte in einem der genannten Layer verschwunden sein. Im Netzwerktreiber des Hypervisors. Oder im Kernel des virtuellen Servers. Oder das erste SDN war falsch konfiguriert.

Containerplattformen und Multitools

Name	OpenShift[1]	Rancher[1]	Pivotal PKS[1]	ICP[1]	Twistlock[2]	AquaSec[2]	NeuVector[2]	Sysdig/Secure/Falco[2]	Cavirin[2]	StackRox[2]
Website	www.openshift.com	rancher.com	pivotal.io/platform/pivotal-container-service	www.ibm.com/cloud/private	www.twistlock.com	www.aquasec.com	neuvector.com	sysdig.com	www.cavirin.com	www.stackrox.com
Basislizenz	Apache 2.0	Apache 2.0	MIT	Apache 2.0	closed	closed	closed	Apache 2.0	closed	closed
Enterprise-Lizenz	ja	ja	ja	ja	only	only	only	ja	ja	ja

[1] Kubernetes-Betriebsplattform; [2] Multitool

Image-Scanner

Name	Anchore	CoreOS Clair	Grafeas / Kritis	OpenSCAP	Black Duck OpsSight	Tenable.io® Container Security
Web	anchore.com	github.com/coreos/clair	github.com/grafeas/	www.open-scap.org	www.blackduck software.com/products/opssight	www.tenable.com/products/tenable-io/container-security
Basislizenz	Apache 2.0	Apache 2.0	Apache 2.0	LGPL 2.1	closed	closed
Enerprise-Lizenz	ja	nein	nein	nein	ja	ja

Oder das Routing der virtuellen Interfaces auf den virtuellen Servern war falsch. Oder da war eine Firewall-Regel falsch gesetzt. Oder das CNI hat einen Bug. Oder die Anwendung hat einen Fehler gemacht. Vielleicht hat auch das Netzwerkkabel einen Wackelkontakt. Oder der Switch einen Fehler bei der CRC-Berechnung der Layer-0-Frames nur bei manchen Paketen. An Letzterem suchte der Autor mal drei Tage. Einen Anwender zu finden, der zwei Switchports in seinem Büro mit einem Hub miteinander verband, war relativ einfach dagegen.

Und so geht es dann weiter. Warum ist der Container nicht gestartet und warum ist er nach 14 Sekunden abgestürzt? Warum starten die Images, die gestern noch einwandfrei liefen, heute nicht mehr ordentlich und benötigen 32 Sekunden lang 100 % CPU? Und warum startet die agile Skalierung dann einen virtuellen Server nach dem anderen, die alle 32 Sekunden zum Booten brauchen und dann sehr schnell sehr teuer sind?

Die Einfachheit der Anwendungspaketierung wird mit einer hohen Komplexität beim Betrieb erkauft. Das Tooling ist komplex und automatisiert. Routineaufgaben erledigt heute ein Kubernetes-Operator. In den 1990ern gab es den Spruch: „Ich werde dich durch ein Shellskript ersetzen.“ Der ist wahr geworden: Kubernetes und seine Werkzeuge ersetzen viele Administratoren. Wer nicht bereit oder fähig ist, sich dieser schönen neuen Welt anzupassen und sein Handwerk neu zu lernen, wird sich bald beim Tonerwechseln im Großraumbüro ertappen. Der Serverraum gehört jetzt Kubernetes und den Admins, die auch programmieren können und das Tooling durchschauen. Linux von CD konnten viele. Kubernetes-Troubleshooting beherrscht nur noch ein kleiner Kreis. Auch hier ist der Wettbewerb hart.

Monolithen und Microservices

Komplexität ist auch bei diesem Thema die Titelmelodie. Auch wenn dies nicht direkt mit Kubernetes zu tun hat, so ist es doch eng verwoben, auch weil Kubernetes selbst eher ein Microservice-Framework ist, eine Sammlung kleiner Dienste. Alle wollen weg von monolithischen Anwendungen, wo ein großer Haufen Code schlecht zu warten und schlecht zu skalieren ist. Also wird das, was früher ein Modul im Code war, in einen eigenen Prozess ausgelagert, der nun als Container in einem oder mehreren Kubernetes-Clustern läuft. Wo also früher der Administrator nur ein Stück Code in einer Datei in einem Anwendungsserver sah und verstehen musste, sieht er nun viele kleine Dienste, die alle miteinander reden wollen und sich auch gegenseitig authentifizieren müssen. Sie stellen alle ihre eigenen Anforderungen an den Betrieb, an den Speicher, an die Überwachung. Der Administrator muss also die Architektur der Software verstehen. Zusätzlich zu seinem Verständnis der Architektur der Betriebsumgebung, die wie oben erwähnt auch nicht einfacher wurde.

Dazu kommen dann noch alle Ansätze für die Skalierung, die erst mit dem Cloud-native wirklich durchstarten. Die Rede ist von verteilten Anwendungen. Das können entweder Komponenten von Microservices sein, die getrennt voneinander skalieren, oder komplett neue Arten, wie man Daten speichert und verarbeitet. Sharded ist hier das Zauberwort. Wo früher ein Datenbankserver als Monolith alle Daten auf einem Datenspeicher ablegte, regieren heute verteilte Datenbanken, die in den meisten Fällen auf den Ideen von Google aufbauen. Von Hadoop über Kafka bis Cassandra sehen wir überall das gleiche Bild: Viele kleine Server verbinden sich zu einem Anwendungscluster, in dem alle Mitglieder gleichberechtigt sind und sich mit Abstimmungsalgorithmen wie Raft über die aktuelle Konfiguration austauschen. Gewählt wird der Anführer, der das zentrale Register der Daten verwaltet, aber nicht alle Daten selbst speichert. Alle im Verbund speichern und reden mit den Clients. Sicherheit vor Verlust wird dabei über Replizierung gewährleistet, der Verbund selbst benötigt also keine ausfallsichere Hardware, keine redundanten Plattenstapel mit RAID 6 und keine doppelten Netzteile mehr. Der einzelne Server kann ausfallen, der Verbund bleibt bestehen, und weder gehen Daten verloren, noch merkt der Anwender etwas vom Ausfall.

Site Reliability Engineering / DevOps

Ausfälle verhindern ist daher auch das von Google entwickelte Betriebsmodell Site Reliability Engineering (SRE), das man auch „DevOps done Right“ nennen könnte. DevOps ist die Kombination zweier früher klar getrennter Laufbahnen und Berufe. Auf der einen Seite gab es die Administratoren, die Herren der Server und Netze. Und auf der anderen Seite waren die Anwendungsentwickler, die diese Server und Netze für ihre Zwecke nutzen, manche sagten auch missbrauchen. Zusammen sind sie Dev und Ops. DevOps definiert die Zusammenarbeit – heute enger als früher. Die beiden Teams, die bei Firmen gerne auch in unterschiedliche Gebäude oder gar in eigenständige Firmen ausgelagert waren, sind nun eines – mit einer gemeinsamen Verantwortung für die Entwicklung und den Betrieb der Anwendungen.

Wo DevOps die Fragen stellt, gibt SRE die Antworten. Ein gutes Beispiel ist die Erfindung des Error-Budgets. Wenn die Verantwortung geteilt wird, ist es erforderlich, diese auch zu messen. Also wird nicht mehr wie früher die Verfügbarkeit der gesamten Anwendung gemessen und im SLA (Service Level Agreement) mit dem Kunden zusammen festgelegt, sondern die Zahl und die Geschwindigkeit der Anfragen geht im SRE-Monitoring in die Indikatoren Service Level Indicator (SLI) und Service Level Objective (SLO) ein, die das DevOps-Team dann zusammen im SLA mit dem meist internen Kunden festlegt. Aber welche Abweichungen sind jetzt in Ordnung, wenn fast 100 % Verfügbarkeit der Anwendung dank verteiltem Hosting schon quasi der Standard ist? Hier kommen die Error-Budgets ins Spiel. Das SLA legt beispielsweise fest, dass 99,5 % der Anfragen erfolgreich sein müssen. Erfolgreich heißt in diesem Kontext: „Der Anwender oder die Anwendung bekommt den angefragten Inhalt und die Antwort erfolgte schnell genug.“ Technisch ausgedrückt heißt das: kein HTTP-Ergebnis-Code aus der 5xx-Serie und die Antwort innerhalb von 80 Millisekunden. Da mittlerweile selbst der Massenspeicher, sprich die Festplatten, über HTTP antworten, ist dies eine für fast alle Dienste einheitliche Definition.

Die gemeinsame Verantwortung des DevOps-Teams schlägt sich so in messbaren Zahlen nieder. Wenn also eine Anwendung in einem Quartal ihr Error-Budget nach einem Monat aufgebraucht hat, dürfen keine neuen Features in die Produktion gehen oder es muss eine Bugfix-Release geben. Und wenn das Error-Budget wenig angetastet ist, ist es möglich, neue Features auszurollen. Die Zahl der Features und das Einhalten des SLA sind dann wieder messbare Größen, die in die Gehälter der Mitarbeiter einfließen können. So kann DevOps nicht nur eine Ausrede für agiles Projektmanagement sein, sondern tatsächlich eine gemeinsame Aufgabe aller.

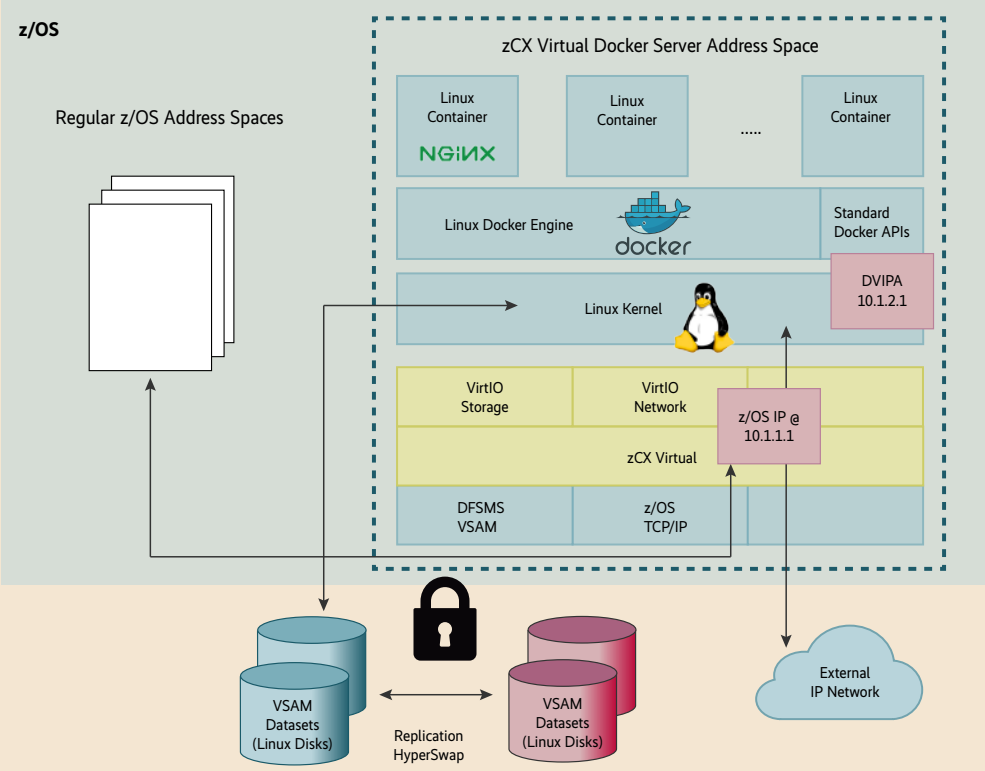
Diese gemeinsame Verantwortung ist aus mehreren Gründen notwendig. Wie oben geschrieben, sind die Cloud-native-Stacks und die Microservices zu sehr miteinander verwoben, als dass die Administration diese alleine betreiben könnte. Und auch der Betrieb der Server und Anwendungen ist heute mehr das Programmieren von Automatismen. Everything as Code, und das gemeinsam in einer Quellcodeverwaltung für die Tausenden Zeilen Konfiguration, ist das Schlagwort.

CI/CD-Tooling

Früher bezeichnete man manuelle Tätigkeiten – etwa ein SRE – als Toil, also sich wiederholende Arbeit. Installation des Switch, des Servers, der Dienste und der Anwendung waren repetitiv, bei guter Organisation auch in festgelegten Abläufen, oft genug frei Hand nach Tagesform des Administrators. Das Ziel ist, den Toil zu minimieren und sich dabei bewusst zu sein, dass die Nirvana Fallacy auch hier gilt: 100 % ist das Ziel, jedoch ein asymptotisches. Man kann sich der perfekten Lösung nähern, sie aber nie erreichen. Auf Arbeitszeit umgelegt ist 50 % Toil und 50 % Programmierung für Administratoren auch bei SRE und DevOps auch in großen Firmen ein guter Wert.

Unterstützung bekommt das Team vom CI/CD-Tooling, also den Werkzeugen, die für Continuous Integration und Continuous Deployment zuständig sind. Die Werkzeugkiste umfasst dabei immer eine Verwaltung für den Anwendungscode und den Code zur Konfiguration der Server und Netzwerke, die Softwarepakete inklusive Betriebssystem und den Shellskript-Server (Runner). Ja, auch wenn Kubernetes selbst deskriptiv arbeitet, also die Administration daraus besteht, den Sollzustand zu beschreiben und bei der Umsetzung des Istzustands zum gewünschten Zustand zuzusehen, so hat sich hier das gute alte Shellskript eingeschlichen. Konkret sieht dies so aus, dass das Team in der Verwaltung des Codes Trigger definiert und diese beim Auslösen ein Shellskript starten. Neumodisch heißt das dann Pipeline. Sprich: Das Skript steuert die Wanderung des Codes durch Tests bis hin zur Installation in einer Betriebsumgebung. Wer diesen Prozess gut im Griff hat, kann so bei einem Commit in den Master Branch im GIT erst die Qualitätstests, dann die Sicherheitstests, dann die Funktionstests und dann das Ausrollen in die Produktionsumgebung vollständig automatisieren. Wenn gewünscht, wird dann bei Fehlern in der Produktion automatisch auf die alte Version zurückgegangen. A-B-Testing und Canary Rollouts sind hier die Schlagwörter.

Wie in jedem Teil des Cloud-native gibt es hier natürlich eine große Auswahl an Open-Source-Produkten, die auch teilweise Enterprise-Lizenzierung ermöglichen. Bekannte



Die z Container Extensions (zCX) in einer logischen Partition (LPAR) als Beispiel für die EAL5+-zertifizierte Betriebsumgebung auf zOS. Gut erkennbar ist der Aufbau eines Node, der sich auf allen Containerhosts auch außerhalb der Großrechner wiederholt (Abb. 2).

Vertreter sind Harbor, das mit Jenkins als Runner arbeitet, und GitLab, das den Runner bereits enthält. Mit Letzterem beschäftigt sich der Artikel „Container legende Wollmichsau“ in dieser Ausgabe ausführlicher.

Sicherheit

Die Sicherheit der Container wird gerne diskutiert. Sie haben sich aus chroot entwickelt, was immerhin eine kleine, aber durchaus nützliche und effektive Sicherheitsmaßnahme war. Google hat 2004 mit cgroups das Konzept der Namespaces auf solide Füße gestellt und den Code der Linux Foundation geschenkt. Nur durch diese Schenkung sind die Container, wie wir sie heute kennen, überhaupt möglich geworden.

Wenn nun der Vergleich der Sicherheitszonen und deren Trennung ansteht, sei ein Verweis auf die EAL5+-zertifizierten LPAR von IBMs System z erlaubt, die mit über 40 Jahren Erfahrung bei der Trennung von Anwendungen den Gold-Standard darstellen (siehe Artikel „Containerfresser“ in der Ausgabe *iX* 1/2020, S. 100). So hoch zertifiziert ist kein Hypervisor oder anderes Werkzeug zum Separieren jemals gewesen und wird es wohl auch nicht werden. Gängig ist es, für unterschiedliche Anwendungen eine Separierung der Hardware zu fordern, da viele Angriffe auch die Grenzen der Typ-1-Hypervisoren überspringen können. Meltdown und Spectre haben hier unangenehm belegt, dass, wenn Anwendungen auf einer CPU oder einem Kern dieser CPU laufen, auch eine Angriffsmöglichkeit besteht. Gleiches hat RowHammer (1 und 2) für RAM aufgezeigt.

Nun muss nicht jede Betriebsumgebung die Anforderung der militärischen Standards für die Multi-Level-Security erfüllen. Für manche ist die Trennung in Container und damit cgroups schon ausreichend und zumindest eine Verbesserung zum gleichzeitigen Betreiben der Anwendungen auf einem Server ohne getrennte Namespaces für Netzwerk, Dateisystem und so weiter. Kubernetes hilft hier mit wiederum Namespace genannten Gruppen von Containern, für die dann Sicherheitsregeln gelten. Auch kann es Container oder Namespaces an bestimmte Containerhosts binden, sodass alle Anwendungen in einem Namespace nur auf einer Gruppe von Hosts laufen und so hardwarenahe Angriffe wie Spectre nur innerhalb des Namespace möglich sind. Dies ist allerdings nur effektiv, wenn auch der Hypervisor die Hosts an die CPU bindet, wie dies etwa VMware seit Version 6.5 automatisch macht.

Hier den richtigen Mittelweg zu finden, ist für jede Organisation eine Herausforderung, für die es zwar einige Beispiele, aber kaum eine belastbare Anforderung aus einem der Sicherheitsstandards gibt. Auch der Baustein „SYS.1.6 Container“ aus dem IT-Grundschutz des BSI gibt hier wenig Hilfestellung.

Fazit

Die Komplexität schreckt viele. Manche wünschen sich auch die einfachere Welt von zum Beispiel Swarm zurück. Dennoch hat Kubernetes in allen Disziplinen des Betriebs von Anwendungen die Führungsrolle übernommen und mit einem Releasezyklus von drei Monaten gibt es weiterhin das Tempo vor, mit dem die Werkzeuge rund um den Cloud-native-Stack und auch die Administratoren der DevOps-Teams mithalten müssen. Die Innovation ist hier durchaus eine Belastung für den stabilen Betrieb, da nicht nur die Software stabil sein muss, auch die Menschen müssen mit diesen neuen Möglichkeiten und Herangehensweisen vertraut sein. In die Köpfe der Mitarbeiter zu investieren, ist daher heute noch wichtiger, als es dies früher war. (avr@ix.de)

Christoph Puppe
ist Security Consultant bei SVA, Auditteamleiter für ISO 27001 nach Grundschutz, Mitautor des Grundschutz-Kompendiums und ehemaliger Penetrationstester.