ownCloud Infinite Scale mit Microservice-Architektur

Mit mehr als zehn Jahren Praxiserfahrung hat ownCloud seine als Dropbox-Alternative gestartete Software grundlegend überarbeitet. Für ownCloud Infinite Scale verspricht man, die Leistungsgrenzen der Plattform zu verschieben.

Von Dr. Udo Seidel

-tract

- ownCloud ist mit seiner als Dropbox-Alternative gestarteten Software seit mehr als zehn Jahren im Unternehmens- und Behördenumfeld aktiv.
- Die zunehmend hinderlichen Leistungsgrenzen von PHP initiierten einen grundlegenden Umbau der Plattform vom LAMP-Stack zur in Go geschriebenen Microservice-Architektur.
- Dank der Zusammenarbeit unter anderem mit dem CERN ist die Software für kommende Anforderungen im Cloud-Umfeld gut gerüstet.

Vor über zehn Jahren trat ownCloud mit der gleichnamigen Datenaustauschplattform als LAMP-basierte – wobei das P hier für PHP steht – Alternative zu Dropbox an. Mittlerweile hat der Nürnberger Anbieter auch ein DSGVO-konformes SaaS-Angebot im Portfolio und konnte sich in den Jahren vor der Pandemie als zentrale Cloud-Software im Bereich Datenaustausch für die bayerischen Kommunen etablieren. Dabei zeigte sich, dass die Architektur in großen Umgebungen an die Grenzen der PHP-Leistungsfähigkeit stieß. Also zogen die ownCloud-Entwickler und -Architekten Bilanz: Was hatte sich in den letzten knapp 10 Jahren bewährt? Was passt nicht mehr so richtig? Welche Trends gilt es zu beachten? Das war die Geburtsstunde von

ownCloud Infinite Scale – kurz oCIS. iX hat sich die Plattform genauer angesehen.

Ein neues Entwicklungsmodell

Für die Anwendungsarchitektur der oCIS-Plattform hat ownCloud quasi auf der grünen Wiese angefangen – allerdings ohne die Erfahrungen oder die Neuentwicklungen des letzten Jahrzehnts zu ignorieren. Der erste oCIS-Git-Commit stammt aus dem August 2019. Anstelle des recht „gemütlichen“ PHP wechselte man auf Go als Programmiersprache. Diese erfreut sich nicht nur bei Cloud-Enthusiasten großer Beliebtheit, sondern verspricht auch einen deutlichen Leistungszuwachs. Eine weitere technische Neuerung ist der Wegfall einer zentralen (relationalen) Datenbank. Im Cloud-Umfeld ist das fast ein natürlicher Schritt.

Die dritte wesentliche Veränderung ist das Entwicklungsmodell hinter oCIS. Sie hat sogar zwei verschiedene Dimensionen: eine organisatorische und eine technische. ownCloud setzt jetzt mehr auf Zusammenarbeit und Kooperation. So basiert die Datenspeicheranbindung auf EOS Open Storage, einem am CERN für den Einsatz beim LHC entstandenen Projekt (siehe ix.de/zsub). Eine weitere Kooperation betrifft eine freie Alternative zur Microsoft-Graph-Schnittstelle. Sie ist REST-basiert und erlaubt einen einfachen Zugriff auf die Daten verschiedener Azure-Cloud-Dienste auf Grundlage der Identität des Benutzers. Zusammen mit der Firma Kopano hat ownCloud das Projekt Libregraph ins Leben gerufen und nutzt dies in oCIS (siehe ix.de/zsub).

Dies ist aber nur die Spitze des Eisbergs. Ausgehend vom LAMP-basierten Produkt gibt es 20 dokumentierte Architekturentscheidungen, die wegweisend für die Entwicklung von oCIS waren und sind. Das Resultat ist ein Produkt mit einer Drei-Schichten-Architektur, basierend auf Microservices. Wer die Entwicklung vielleicht schon ein oder zwei Jahre beobachtet hat: Da gab es alle paar Wochen eine neue

technische Vorschau. Etwas untypisch im traditionellen Open-Source-Umfeld zierte die Hauptversion schon damals eine 1 und keine 0. Im November 2022 erschien dann (konsequenterweise) oCIS 2.0.0 – reif für den produktiven Einsatz.

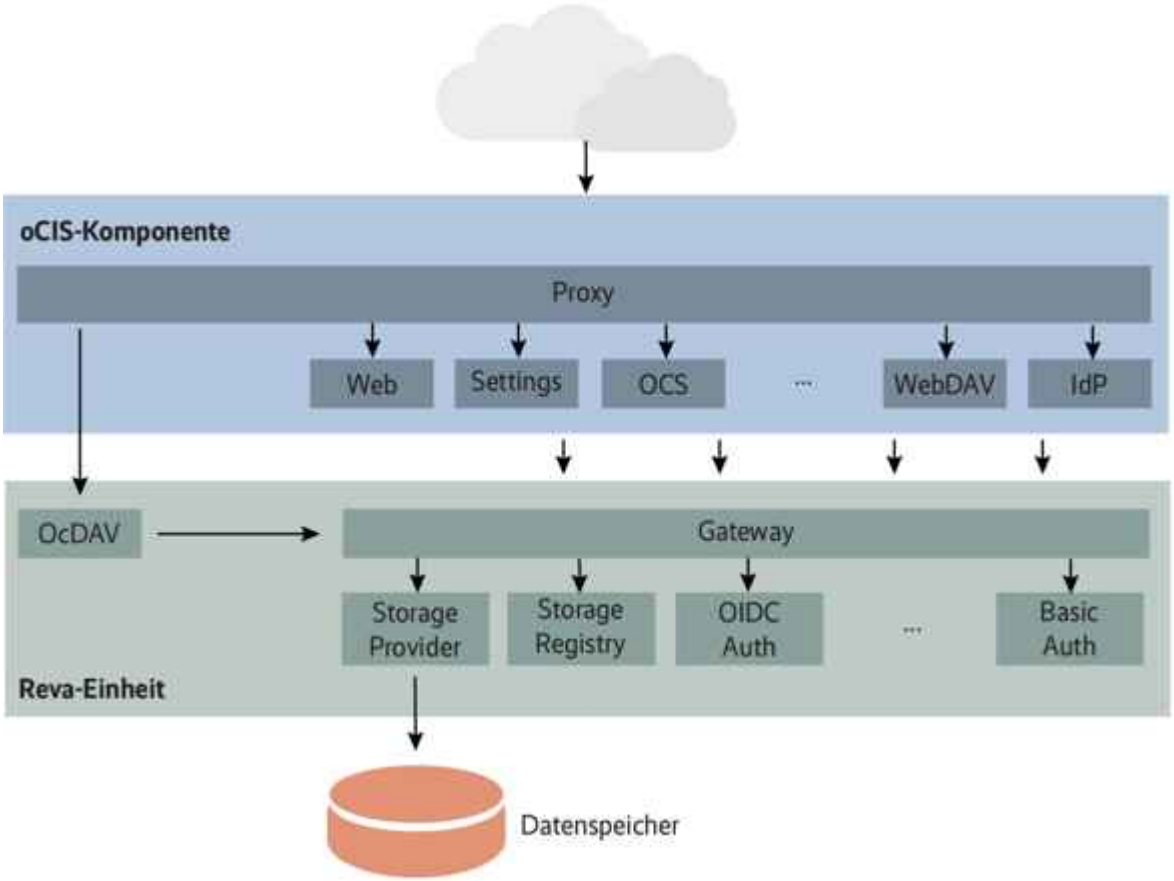
Die Software steht unter der Apache-2.0-Lizenz und läuft auf Linux und macOS. Ersteres in 32 und 64 Bit sowohl für x86 als auch für ARM. Es gibt Clients für iOS, Android, Windows, macOS und Linux sowie eine Weboberfläche für den Zugriff über einen Webbrowser. oCIS verpackt die Software in eine nicht einmal 90 MByte große Binärdatei. Für erste Schritte sind keine großen Hardwareanschaffungen nötig: Beim Hauptspeicher reichen 256 MByte, für produktive Umgebungen sollte man aber mindestens 4 GByte einplanen. Bezüglich CPU und Netzwerkbandbreite macht ownCloud keine konkreten Vorgaben. Die tatsächlichen Anforderungen variieren je nach Anwendungsszenario zu sehr, bei der Netzwerkbandbreite spielen die Anzahl der Clients, der Veränderungen und deren Größe eine wesentliche Rolle.

Ohne weitere Unterstützung durch ownCloud sind Installation und Betrieb von oCIS quasi kostenfrei. Nur Support vom Hersteller, Zugriff auf das Kundenportal und auch Funktionen wie Clients mit dem eigenen Firmenlogo sind mit Kosten verbunden. Die Konditionen orientieren sich an der Benutzeranzahl mit skalierendem Discount bei großen Umgebungen.

Von ganz oben betrachtet

Grob betrachtet besteht oCIS aus drei Komponenten. Da ist zunächst der in unterschiedlichen Inkarnationen vorliegende Client. Der oCIS-Server-Teil besteht aus zwei Hauptkomponenten: Benutzer- und Datenverwaltung. Zum Authentifizieren der Benutzer greift oCIS auf OpenID Connect (OIDC, siehe ix.de/zsub) zurück. Damit kann die Software alle mit diesem Protokoll kompatiblen Identity Provider (IDP) verwenden. Im Cloud-Umfeld ist dieser Ansatz inzwischen der De-facto-Standard. Im einfachsten Fall kann hier ein LDAP-

Server einspringen. Genau genommen ist dieser ein Identity Management System (IDM). Der LDAP-Server muss dabei nicht dediziert für oCIS installiert sein. Für die allerersten Schritte bringt oCIS einen kleinen IDP mit, der auf dem bereits erwähnten Libregraph fußt. Für das Kennenlernen der Software mithilfe lokaler Benutzer reicht das vielleicht schon. Ein realistischer Einsatz, bei dem auch Benutzer mehrerer Firmen zusammenarbeiten, sollte unbedingt OIDC verwenden.



Für oCIS setzt ownCloud auf eine dreischichtige Microservice-Architektur (Abb. 1).
Bei der Datenverwaltung gibt es zunächst die Datenträger selbst. Das können lokale Platten in einem Rechner sein, aber auch netzwerkbasierter Storage wie Amazon S3 oder NFSv4 ist möglich. Der Zugriff darauf erfolgt über Treiber. Intern verwaltet oCIS die Speicher in sogenannten Spaces, die es in einer Registry vermerkt. Die Space Registry ist der Einstieg in die unteren Schichten des oCIS-Storage-Stacks. Sie verwaltet den Namensraum der Benutzer, sprich: Welche Daten

dürfen sie lesen und/oder schreiben? Über die registrierten Spaces und Treiber erfolgt dann der eigentliche Zugriff. Im Detail ist die Datenverwaltung etwas komplexer – dazu später mehr.

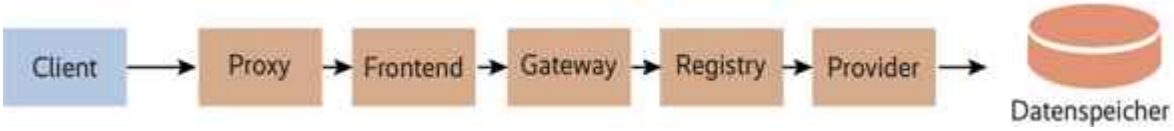
Für die internen Netzwerkverbindungen verwendet oCIS fast ausschließlich das von Google initiierte RPC-Framework gRPC. Durch den Microservice-Ansatz ist das nicht weiter verwunderlich. Damit geht einher, dass oCIS für sich eine ganze Reihe von Ports beansprucht. Wer für die Zukunft gewappnet sein will, sollte alle zwischen 9000 bis 10 000 schon mal reservieren. Im Moment scheint sich aber alles im Bereich bis 9300 abzuspielen. Neben gRPC kommt natürlich auch LDAP für die Benutzerauthentifizierung zum Einsatz. Zur einfacheren Netzwerkkommunikation verwendet oCIS intern ein Gateway. Es dient als Pförtner zu den für die Datenverwaltung zuständigen Microservices.

Die Kommunikation nach außen teilt oCIS in zwei Kategorien. Für externe Datenspeicher kommen Protokolle wie S3 oder NFSv4 zum Einsatz, die oCIS-Clients wiederum benutzen das ownCloud-eigene Synchronisierungsprotokoll auf WebDAV-Basis. Dabei ist hier eine Art API-Gateway vorgeschaltet, das als Eingang zu den Microservices für Benutzerauthentifizierung, den Einstellungen oder den ownCloud-APIs dient.

Vom Ästlein zum Zweiglein

Mit oCIS haben die ownCloud-Entwickler eine Microservice-Struktur eingeführt, die auf den ersten Blick etwas überwältigt. Die Software landet als eine Binärdatei auf dem System und startet im Vollausbau weit über 20 Dienste. Die gehören zu drei Gruppen: Benutzerverwaltung, oCIS-Komponenten und die vom CERN stammende Reva-Einheit. Noch mal zur Erinnerung: ein Gateway dient als Schnittstelle zwischen den beiden Letzteren. Zur Reva-Einheit gehören Dienste für die Kommunikation mit den Datenträgern, etwa storage provider. Dazu später mehr. Es gibt auch eine Verbindung von der Reva-

Einheit zur Benutzerverwaltung. Dort sind Identity Provider und Management beheimatet und füttern die Attribute user provider und group provider mit den erforderlichen Daten. Die oCIS-Komponenten kann man als Schaltstelle des Gesamtsystems verstehen. Hier erfolgt die Verarbeitung und Weiterleitung der Clientanfragen. Es gibt jeweils einen Microservice für die verschiedenen Protokolle. Hier findet sich auch die Konfiguration des Systems wie der Detailgrad der Protokolldateien oder der Pfad zu den x509-Zertifikaten für die verschlüsselte Kommunikation.



Dieser exemplarische Arbeitsablauf zeigt das Zusammenspiel der oCIS-Microservices (Abb. 2).
Intern benutzt oCIS das Projekt suture zur dynamischen Verwaltung der Microservices. Auch an anderer Stelle haben die ownCloud-Entwickler das Rad nicht neu erfunden. Als Gerüst für die Implementierung des Microservice-Ansatzes nutzen sie die Ergebnisse des Projektes go-micro (siehe ix.de/zsub). Um unnötige Quellcodekopien zu vermeiden und die vielen Dienste zu vereinheitlichen, kommt die Eigenentwicklung ocis-pkg zum Einsatz. Sie benutzt das go-micro-Gerüst zur Implementierung der Schnittstellen für Serverkomponenten und Clients.

Der Wechsel von LAMP zur Microservice-Architektur ist den ownCloud-Entwicklern gelungen. Das gilt ebenso für die Verwendung bereits etablierter Software und Komponenten. Die Tatsache, dass oCIS dennoch als eine einzelne Binärdatei daherkommt, erleichtert die Installation, Aktualisierung und Wartung. Die Anzahl der reservierten Ports wirkt allerdings etwas zu großzügig – zumindest in der geplanten Stufe. Das kann insbesondere dann problematisch werden, wenn die einzelnen Microservices auf verschiedenen Rechnern laufen sollen. Momentan ist das nur ein hypothetischer Fall, jedenfalls nach den Installationsrezepten von ownCloud. Generell ist die Verteilung von Microservices auf

verschiedenen Rechner im Cloud-Umfeld nicht unüblich. Dann müssen die Entwickler auch über die Verschlüsselung der gRPC-Verbindungen nachdenken. Und eigentlich ist die Antwort offensichtlich: ein Service Mesh. Hier gibt es verschiedene Implementierungen und Anbieter.

Hinter den Datenkulissen

Für das Herzstück, die Datenverwaltung zum Verteilen von Dateien und Verzeichnissen, haben sich die ownCloud-Entwickler für das ebenfalls am CERN beheimatete Projekt Reva entschieden (siehe ix.de/zsub). Das ist eine Referenzimplementierung des CS3-Schnittstellenstandards (CS3 – Cloud Storage Services for Synchronization and Sharing). Die Nutzung von Reva ist eine der oCIS-Architekturentscheidungen. Ein wichtiger Punkt war hier auch die Implementierung eines verteilten und hochverfügbaren Systems zum dauerhaften Speichern von Benutzerinformationen. In traditionellen ownCloud-Installationen war dies nicht möglich oder hätte signifikanten Entwicklungsaufwand erfordert.

Wie erwähnt kann oCIS sowohl auf lokale Datenträger als auch auf Storage im Netz schreiben. Dabei abstrahiert die Software noch mal über eine weitere Schicht: Decomposed FS. Die Kernidee dieses Dateisystems ist die Dateiverwaltung anhand von UUIDs und eine für Nutzer nicht sichtbare sowie recht flache Struktur der Datenablage. Nach außen zeigt oCIS die gewohnte Verzeichnisstruktur mit Dateien. Im Hintergrund findet oCIS die entsprechenden Daten aber nicht per Pfad, sondern per UUID. Über Treiber greift Decomposed FS auf die eigentlichen Datenträger zu.

Lokale Platten müssen ein POSIX-Dateisystem haben, die Entwickler empfehlen XFS oder ZFS – mit Einschränkungen funktioniert aber auch ext4. Ein wichtiges Merkmal der POSIX-Dateisysteme ist die verfügbare Speichergröße für erweiterte Attribute. Wie Listing 1 zeigt, speichert Decomposed FS dort viele Verwaltungsinformationen. In großen Installationen mit

ext4 würde oCIS dann künstlich limitiert. Bei den Netzwerkspeichern ist die Situation komplizierter. Prinzipiell funktioniert oCIS mit NFS, S3, CephFS oder sogar EOS. Für den produktiven Einsatz ist bislang aber nur Amazon S3 freigegeben. Es ist zu erwarten, dass die anderen nachfolgen. Wer jetzt schon mal reinschnuppern will, findet bei ownCloud gute Dokumentation dafür.

Listing 1: Ausgabe von getfattr -d zeigt Extended-Attribute

```
# file: storage/users/spaces/4c/510ada-c86b-4815-8820-42cdf82c3d51/nodes/4c/[…]
user.ocis.blobid=""
user.ocis.blobsize="0"
user.ocis.name="Albert Einstein"
user.ocis.owner.id="4c510ada-c86b-4815-8820-42cdf82c3d51"
user.ocis.owner.idp="https://localhost:9200"
user.ocis.owner.type="primary"
user.ocis.parentid=""
user.ocis.propagation="1"
user.ocis.space.alias="personal/einstein"
user.ocis.space.name="Albert Einstein"
user.ocis.space.type="personal"
```

Abstraktion ist ein wiederkehrendes Motiv bei der oCIS-Datenverwaltung. Das kann beim Lesen der Dokumentation verwirrend sein. Allerdings bleibt Anwendern diese Komplexität verborgen. Nur wer es genau wissen will, muss sich mit Begriffen wie Storage Space oder Space Registry herumschlagen. Da eine ausführliche Darstellung den Artikelumfang sprengen würde, folgt hier nur eine kurze Darstellung. Storage Space ist eine logische Zusammenfassung von Dateien und Verzeichnissen. Das könnten etwa die Daten eines Projektes oder einer Abteilung sein. Wichtig ist, dass es ausschließlich einen logischen Besitzer der Daten gibt. Dieser Storage Space ist mit einer eindeutigen Nummer versehen.

Die nächste Ebene nennt sich Storage Provider. Sie verwaltet

einen oder mehrere Storage Spaces. Dabei ist diese Zuordnung zwar immer eindeutig, aber veränderbar. Ein Storage Space kann von einem Provider zu einem anderen wechseln. Der Storage Provider ist außerdem für den eigentlichen Zugriff auf die physischen Datenspeicher zuständig. Dabei kommen die oben beschriebenen Treiber zum Einsatz. Der Storage Provider teilt sich die Verwaltungsaufgaben mit der Space Registry. Letztere kümmert sich um Dinge wie freien Platz, Benutzerzugehörigkeit oder auch Quotas. Hier ist der Namensraum angesiedelt, der entscheidet, ob der Anwender Zugriff auf die Daten hat oder nicht. Der Storage Provider ist auf der unteren Schicht des Stacks unterwegs, dem Datenzugriff auf den eigentlichen Storage.

Zum Abschluss noch zwei Hinweise. Bevor eine Benutzeranfrage beim Storage Provider ankommt, durchläuft sie einige Stationen beziehungsweise Microservices. Das fängt beim oCIS-Proxy an und führt am Ende zum Gateway, das oben als Pförtner in die Reva-Welt beschrieben wurde. Wen es genauer interessiert, der kann den genauen Datenfluss in der ownCloud-Dokumentation nachlesen (siehe ix.de/zsub). Hinweis Nummer zwei ist eher praktischer Natur: Wer seine Daten von einer anderen Plattform in oCIS migrieren möchte, kann auf das Werkzeug Rclone zurückgreifen.

Potenzielle Installationsszenarien

Der Start mit oCIS unter Laborbedingungen ist denkbar einfach (siehe Kasten „Los gehts“). Auch die Integration in das Hoch- beziehungsweise Herunterfahren von Linux lässt sich ganz einfach bewerkstelligen. Die Dokumentation enthält eine detaillierte Anleitung inklusive systemd-Konfiguration. Das Gleiche gilt für die Benutzer, die auf dem Containerpfad unterwegs sind, sowohl mit als auch ohne Orchestrierung durch Kubernetes. Läuft schon ein entsprechender Cluster, lässt sich oCIS recht schnell mit den Helm Charts des Projektes aufsetzen. Container und Kubernetes sind laut Aussage des

Projektes die bevorzugte Plattform. Da verwundert es nicht, dass hier quasi alles im Detail schon bereitliegt (siehe ix.de/zsub).

Sehr positiv ist zu vermerken, dass sich für quasi jede Plattform ein Installationsrezept für oCIS findet. Egal, ob echte Hardware, VM, Container mit oder ohne Orchestrierung: Jeder kommt auf seine Kosten. Die Entscheidung, die Software als einzelne ausführbare Datei auszuliefern, hat sich hier ausgezahlt. Die Unterstützung von 32-Bit-Plattformen – sowohl auf x86 als auch auf ARM – ist inzwischen auch selten und verdient das Prädikat Luxus. Damit sind die ersten und vielleicht auch zweiten Schritte sehr einfach. Die Bewährungsprobe im harten Produktionsalltag steht aber noch aus.

Los gehts

Die ersten Schritte im Labor mit oCIS sind einfach. Die Software kommt als eine einzige Binärdatei daher. Einfach herunterladen, eventuell umbenennen und sie als ausführbar markieren – und schon kann es losgehen. Der Start erfolgt in zwei Stufen. Zunächst generiert man über das Kommando ocis init eine Konfigurationsdatei. Dabei erzeugt das Programm eine Grundkonfiguration und legt den Admin-Benutzer inklusive Passwort an. Danach lässt sich die oCIS-Instanz mit ocis server starten (siehe Listing 2). In diesem Fall laufen alle mitgelieferten Microservices; der Aufruf ocis list zeigt sie an.

Ein alternativer Weg führt über ein vorgefertigtes Docker-Image von ownCloud (siehe ix.de/zsub). Auch hier ist der Start zweiphasig. Zunächst startet man die Containerinstanz und gibt das Kommando ocis init mit auf den Weg. Danach kommt dann das Hochfahren der oCIS-Instanz. Dabei gilt es, den Port 9200 für den Zugriff auf das Webfrontend entsprechend umzuleiten:

```
docker run --rm -p 9200:9200 -v ocis-config:/etc/ocis -v ocis-data:/var/lib/ocis owncloud/ocis
```

Und es geht sogar noch einfacher. Wer nur mal in oCIS reinschauen möchte, kann die Onlinedemo benutzen (siehe ix.de/zsub). Die notwendigen Log-in-Daten stehen auf der Startseite.

Listing 2: Fünf Schritte zur ersten oCIS-Instanz

```
$ wget -nd https://download.owncloud.com/ocis/ocis/stable/2.0.0/ocis-2.0.0-linux-amd64
...
$ mv ocis-2.0.0-linux-amd64 ocis
$ chmod 755 ocis
$ ./ocis init
...
==========================================
generated OCIS Config
==========================================
configpath : /home/ocis/.ocis/config/ocis.yaml
user : admin
password : 5p2ZjjGuiI#rVRF*P0SHpU+KKzu$&Dnv
$ OCIS_INSECURE=true IDM_CREATE_DEMO_USERS=true PROXY_HTTP_ADDR=0.0.0.0:9200 OCIS_URL=https://localhost:9200 ./ocis server
```

Im Produktionsbetrieb stellt sich immer die Frage nach Skalierbarkeit, also einer hohen Verfügbarkeit und ausreichender Leistungsfähigkeit des Dienstes. Bei oCIS gilt es, dafür drei Fragen zu beantworten. Welche Microservices dürfen parallel laufen? Wie halte ich die Informationen der verschiedenen Instanzen synchron? Wie verteile ich die Last, ohne Duplikate oder Lücken zu erzeugen? Zur ersten dieser Fragen finden sich ausreichend Informationen in der Dokumentation. Sie beschreibt unter anderem, welche Dienste nur einmal laufen dürfen. Außerdem zeigt ein genauerer Blick in die Kubernetes Helm Charts, wo horizontale Skalierung einfach möglich ist. Das oCIS-Binary enthält alle Microservices und man entscheidet per Kommandozeilenoption, welche tatsächlich starten sollen.

oCIS kann sowohl vertikal als auch horizontal skalieren – typischerweise, wenn es um die Erweiterung des verfügbaren Datenspeichers geht. Sprich: Weitere Rechner mit lokalen Datenträgern treten dem oCIS-Verbund bei. Analoges gilt, wenn die Netzwerkbandbreite für die Clients an ihre Grenzen stößt.

Unbeantwortet ist dabei die Frage der Datenverteilung und -synchronisation auf den Speichermedien. Haben alle oCIS-Instanzen Zugriff auf alle Daten? Gibt es eine Partitionierung/Unterteilung? Wenn ja, wie wissen die oCIS-Microservices, wer wofür zuständig ist? Hier gibt es keine befriedigende Antwort in der Dokumentation. Natürlich löst ein verteiltes Dateisystem wie CephFS, NFSv4 oder auch ein Cloud-Speicher S3 dieses Problem. Für den produktiven Einsatz ist aber im Moment nur S3 erlaubt.

Auch vertikale Skalierung kann vorkommen. Die oCIS-Instanzen benutzen den RAM als Zwischenspeicher für den Zugriff auf die eigentlichen Daten. Die einfachste Variante ist also die Vergrößerung des Hauptspeichers. Alternativ wäre auch das Verteilen über neue Instanzen möglich, was dann horizontales Skalieren wäre. Weniger Daten, also weniger Belastung des Hauptspeichers.

In der Dokumentation von oCIS enttäuscht jedoch der Abschnitt über Hochverfügbarkeit und Skalierung. Es fehlen klare Rezepte und Handlungsanweisungen. Hier muss ownCloud unbedingt nachbessern. Einmal, weil diese Aspekte essenziell für den seriösen Einsatz im produktiven Umfeld sind. Außerdem wirkt der Bruch im Informationsgehalt der sonst sehr guten Dokumentation etwas befremdlich.

Fazit

Mit oCIS hat ownCloud einen gewaltigen Wechsel seiner Datenverteilungsplattform vollzogen. Da ist der Technologiewechsel von LAMP zu einer Microservice-Architektur mit Go als Programmiersprache. Der geoverteilte und föderative

Ansatz ist modern und den aktuellen Anforderungen entsprechend. Eine wichtige Aufgabe für die nähere Zukunft ist die Unterstützung weiterer Cloud-Speicherdienste und verteilter Dateisysteme für die Datenablage. Die Dokumentation ist insgesamt sehr umfangreich. Die Kooperationen – insbesondere mit dem CERN – geben dem Produkt eine zusätzliche Qualität in Bezug auf Planungssicherheit und Interoperabilität. Die ersten Schritte sind einfach und gut dokumentiert. Die Bewährung im harten Produktionsalltag steht aber noch aus. (avr@ix.de)

1. Quellen
2. Links zu weiteren Hintergrundinformationen und zu Download- und Demo-Seiten: ix.de/zsub

Dr. Udo Seidel

war seit 1996 als Linux-/Unix-Trainer, Administrator, Senior Solution Engineer und Chefarchitekt tätig. Er arbeitet heute als Senior Customer Experience Architect im europäischen Gebiet für Kong Inc.