Functions as a Service

Höchst wolkig

Jan Bundesmann

Bei Stack Overflow ergab eine Umfrage unter den dort aktiven Entwicklern, dass Serverless derzeit höchst angesagt ist, da es erlaubt, sich auf das Dev in DevOps zu konzentrieren. Darin liegt allerdings nicht der einzige Nutzen, Serverless lässt sich auch im Rahmen des Internet of Things oder für Big-Data-Anwendungen verwenden.

Exakte Abrechnung nach Verbrauch und eine Infrastruktur, um die sich externes Personal kümmert – Merkmale von Cloud-Angeboten. In diesem Sinne erfüllt Serverless Computing den Cloud-Anspruch in höchstem Maße. Denn darin vereinen sich die Aspekte Skalierbarkeit und bedarfsgenaue Abrechnung. Gleichzeitig fällt zwar nicht der Server selbst, wohl aber die Administration der Infrastruktur weg. Anwender laden einzelne Funktionen hoch. Die Abrechnung erfolgt nach verwendeten CPU-Zyklen und benötigtem Übertragungsvolumen.

Den Begriff „Serverless“ machte Amazon 2014 mit der Einführung seines Angebots AWS Lambda populär [1]. Das Konzept ist allerdings bereits älter. Iron.io verwendete den Term bereits 2011 (siehe „Alle Links“ am Ende des Artikels). Der Name ist etwas irreführend, sind die Server letztendlich doch nicht wegzudenken. Andere Firmen wählen dafür die Bezeichnung „Functions as a Service“ (FaaS). Auch Webhook-Dienste fallen letztlich in diese Kategorie.

Der Markt präsentiert sich uneinheitlich. Auf der einen Seite steht Manta von Joyent. Dabei handelt es sich primär um einen Objekt-Store. Die gespeicherten Daten können direkt darin verarbeitet werden, wobei Manta ein breites Spektrum an Sprachen und Werkzeugen zur Verfügung stellt. Das andere Ende stellen Dienste wie hook.io oder Fission dar: in erster Linie Empfänger für Signale (Storage-Events, HTTP-Anfragen), die das Ausführen einer Funktion anstoßen. Die Komponente Speicher ist kein Bestandteil mehr, sondern von den Nutzern manuell hinzuzufügen.

Ebenfalls unterschiedlich sind die Bezugsmethoden. Einige Dienste lassen sich aus der Public Cloud beziehen, manche aber auch lokal im eigenen Rechenzentrum installieren. Nur als gehostete Variante gibt es AWS Lambda, Google Functions, hook.io und Webtask.io. Alle anderen gibt es auch für die Private Cloud – allerdings mit unterschiedlichen Schwierigkeitsgraden bei der Installation. Joyents Manta etwa läuft in der hauseigenen Cloud-Infrastruktur Triton, die wiederum im OpenSolaris-Derivat SmartOS zu Hause ist. Azure Functions erfordert den vollen Azure Stack. Fission von Platform9 setzt Kubernetes voraus und ist somit kompatibel zu allen Systemen, die das Microservice-Orchestrierungstool beherbergen können.

AWS Lambda

Den Begriff „Serverless“ trug Amazon mit der Einführung seines Dienstes AWS Lambda in die Öffentlichkeit. Seither

hält sich der Name hartnäckig (siehe Kasten „Wo sind die Server?“), obwohl natürlich die Administratoren der Dienstanbieter sich weiterhin mit Servern auseinandersetzen.

Lambda-Funktionen kennen Programmierer als Funktionen, die nur über Referenzen oder Zeiger angesprochen werden. Sie sind flüchtig und werden zum Beispiel im Zuge von Garbage Collection gelöscht. Functions as a Service verhalten sich in gewisser Weise ähnlich: Sie belegen Ressourcen lediglich während des Aufrufs, danach stehen Speicher und CPU wieder anderen Diensten zur Verfügung. So bietet sich das Konzept für Cloud-Provider an, die ihre Server maximal auslasten wollen.

Bei Amazon verknüpfen Anwender eine Lambda-Funktion zuerst mit einem Trigger. Das kann ein HTTP-Endpoint sein oder ein Event in einem anderen AWS-Dienst. Häufig ziehen Beispiele die Schnittstelle des Objektspeichers S3 heran. Dieser meldet etwa, wenn Dateien hochgeladen wurden, und eine Lambda-Funktion könnte im Fall von Bildern automatisch Thumbnails erstellen. Weitere Trigger können von der Management-Schnittstelle CloudWatch, von DynamoDB oder von Kinesis kommen. Speziell in der Region Europa (Irland) stehen noch Alexa Skills als Trigger zu Auswahl (siehe auch „Sehr gesprächig“ auf S. 44), um ein Backend für Amazons Sprachassistenten Alexa zu erstellen.

Die Funktionen können in Node.js, Java, C# oder Python geschrieben werden. Zur Eingabe gibt es ein Webformular. Alternativ lassen sich ZIP-Archive mit den benötigten Modulen hochladen oder eine Datei aus S3 wählen.

AWS erlaubt eine feinkörnige Rechtevergabe. Darüber lässt sich steuern, auf welche internen Schnittstellen die Lambda-Funktion zugreifen darf, und andererseits, welche Dienste die Funktion aufrufen dürfen.

Wo sind die Server?

Gerade Administratoren könnte der Begriff „Serverless“ sauer aufstoßen, denn Server braucht es auch bei diesem Dienst. Die Kunden kommen damit aber nicht mehr in Berührung. Dass dies auch bei anderen Cloud-Angeboten der Fall ist, ändert nichts an der Tatsache, dass die meisten Nutzer inzwischen wissen, was mit dem Term gemeint ist. Schuld daran dürfte auch sein, dass Branchenführer Amazon sich für den Begriff entschieden hat. Stack Overflow hat in seiner alljährlichen Umfrage gerade festgestellt, dass Serverless als Plattform Platz 2 auf der Beliebtheitsskala erreicht hat. Beliebter ist nur Linux, aber es liegt knapp vor Amazon AWS.

Alternative Bezeichnungen lauten „Functions as a Service“ (FaaS) oder „Event-triggered Microservices“ – die Übersicht zeigt aber auch, dass die Angebote sich nicht beliebig miteinander vertauschen lassen. FaaS ist wahrscheinlich der kleinste gemeinsame Nenner, schließlich müssen Anwender in jedem Fall eine Funktion zum Dienst hochladen, die sie später unter bestimmten Bedingungen aufrufen können. Bei AWS Lambda oder Joyents Manta allerdings ist die Verbindung mit dem jeweiligen Objektspeicher (auf dem dazugehörigen Server) deutlich, während auf der anderen Seite die reinen Webdienste oder die Kubernetes-Applikationen einen eher flüchtigen Charakter haben.

Nutzer können bis zu eine Million Aufrufe und bis zu 400 000 GByte-Sekunden Datenverarbeitungszeit jeden Monat kostenlos nutzen. Dieses Angebot langt eventuell während des Ausprobierens und für Privatanwender. Die Zeitkosten berechnen sich nach dem Arbeitsspeicher, den eine Funktion zugewiesen bekommen hat, und der tatsächlichen Ausführungszeit – hier ist also darauf zu achten, nicht unnötig viel Speicherbedarf zu veranschlagen. Weitere Rechenzeit kostet 0,001667 Cent pro GByte und Sekunde sowie 20 Cent für eine Million Aufrufe. Zusätzlich weist Amazon auf die möglichen Kosten für Datenübertragung und Storage beziehungsweise Datenbankkapazität hin.

Azure Functions

Microsoft integriert einen Funktionendienst über Azure Functions in der Public Cloud. Mit der in Aussicht gestellten Private- und Hybrid-Variante Azure Stack soll sich das Angebot auch lokal installieren lassen.

Einrichten lässt sich eine Azure Funktion über das Azure Dashboard: New | Compute | Function App oder direkt über functions.azure.com. Dieser Prozess läuft in der Regel über das GUI, ein Kommandozeilen-Interface existiert bisher lediglich für Windows, obwohl Microsoft bereits angekündigt hat, es auch für andere Plattformen bereitzustellen. In der Eingabemaske zum Einrichten wählen Anwender zuerst Sprache und Szenario, also die Art der Trigger. Zur Verfügung stehen C#, F#, JavaScript, Bash, Batch, PowerShell, PHP und Python. Die einfachste Steuerung erfolgt über Timer oder über manuellen Aufruf per GUI. Ebenfalls lassen sich Webhooks, HTTP-Requests oder spezielle GitHub-Webhooks wählen. Letzteres zielt auf eine Nutzung als Continuous-Integration-Werkzeug. Blobtrigger reagieren auf das Hinzufügen von Dateien in einen Azure-Storage-Container. Im Dashboard können Azure-Kunden über den Menüpunkt „Internet of Things“ Events definieren, die als Trigger dienen. Außerdem lassen sich Jobs in Queues einfügen.

Microsoft erlaubt nicht alle Kombinationen aus Sprache und Trigger. Python etwa lässt sich nur in experimentellen Szenarios einsetzen. Dazu gehört lediglich der QueueTrigger, also das manuelle Anlegen eines Funktionsaufrufs. Freie Auswahl des Szenarios haben Anwender nur bei C#, F# und JavaScript. Das GUI bietet stets eine ausführliche Dokumentation zur gewählten Eingabemaske.

Aus der Public Cloud kostet das Ausführen der Funktionen 0,0016 Cent pro Sekunde Ausführungszeit und belegtem GByte Arbeitsspeicher. Außerdem verlangt Microsoft 20 Cent je einer Million Aufrufe. Wie üblich ist ein freier Sockel enthalten: pro Monat 400 000 GByte-Sekunden und 1 Million Aufrufe.

iX-TRACT

- Serverless Services oder Functions as a Service bezeichnen ein Cloud-Angebot, bei dem lediglich Funktionscode hinterlegt wird, während die eigentliche Infrastruktur für die Anwender unsichtbar bleibt.
- Neben dem Verstecken der Server ist sekundengenaue Abrechnung ein Merkmal dieser Herangehensweise, was sich gerade im Bereich von Microservices positiv auswirken kann.
- Sind Functions as a Service in ein breites Angebot eingebunden, können Nutzer viele Funktionen und Event-Trigger verwenden, müssen jedoch das Vendor-Lock-in berücksichtigen.

Google Cloud Functions sind sehr übersichtlich beim Einrichten. Wie bei den meisten Diensten wählen Anwender zu Beginn den benötigten Arbeitsspeicher, der später die Grundlage für die Berechnung des Preises bildet (Abb. 1).

Cloud Functions

Google kündigt sein Cloud Functions noch als Betaversion an. Als Sprache versteht es lediglich JavaScript, das in einer Node.js-Umgebung läuft. Der Code lässt sich inline über das Web-Frontend editieren. Alternativ kann man ihn als ZIP-Archiv direkt oder über den Cloud Storage hochladen.

Mögliche Trigger sind Anfragen über eine HTTP-API, Storage Bucket Events und das Google-interne asynchrone Messaging-System Cloud Pub/Sub. Für Letzteres muss man dort ein Topic erstellen und selbiges bei der Definition der Funktion angeben.

Google bietet ein Freikontingent von bis zu 2 Millionen Aufrufen, 400 000 GByte-Sekunden und 200 000 GHz-Sekunden. Der freie ausgehende Traffic beträgt 5 GByte, zu anderen Google-Schnittstellen ist er genauso wie der eingehende Datenverkehr nicht beschränkt. Darüber hinaus verlangt Google 40 Cent pro Million Aufrufen, 0,00025 Cent pro GByte-Sekunde und 0,0001 Cent pro GHz-Sekunde.

Open Whisk

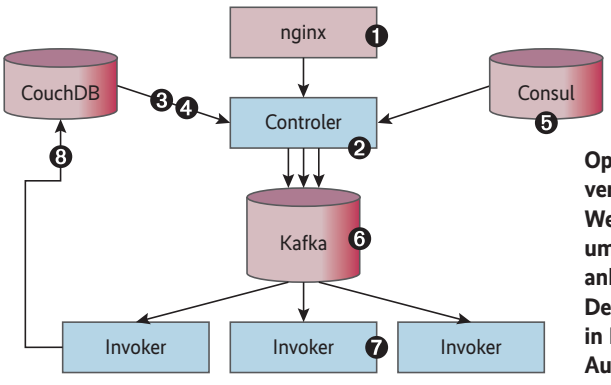
Ursprünglich hatte IBM die Entwicklung von Open Whisk in der Hand, von Anfang an allerdings eine Open-Source-Lizenz für das Projekt gewählt. Inzwischen hat Open Whisk den Status eines Apache-Incubator-Projekts, soll also künftig von der Apache Software Foundation betreut werden.

Beim Design versuchten die Entwickler, viele Aufgaben von gängigen Werkzeugen erledigen zu lassen. nginx bildet die Schnittstelle nach außen und leitet Anfragen an den Controller weiter. Um Authentifizierung und Autorisierung kümmert sich CouchDB. Consul weiß, wo die Dienste liegen, und Kafka übernimmt das Queuing. Das eigentliche Herz von Open Whisk bildet der Invoker. Dieser startet Docker-Container mit der gewünschten Umgebung (derzeit sind Node.js und Swift implementiert) und initiiert die zuvor definierte Funktion, um die Anfrage zu bearbeiten.

Nicht nur bei der Authentifizierung spielt übrigens die Datenbank CouchDB die entscheidende Rolle, sie speichert auch die Funktionen und die Ergebnisse. Definition und Aufrufen einer Aktion kann über das Kommandozeilentool *wsk* erfolgen:

```
wsk action create myAction action.js
wsk action invoke myAction
```

Open Whisk kennt zudem Trigger wie Datei-Uploads oder eingehende Mails. Regeln verbinden diese mit einem Funktionsaufruf, also einer zuvor definierten *action*. Es ist auch möglich, mit einem Trigger mehrere Aktionen zu verbinden. Bilder können etwa nach dem Upload mit einem Wasserzeichen versehen und anschließend in verschiedene Größen umskaliert werden.



Open Whisk setzt verschiedene Open-Source-Werkzeuge zusammen, um Functions as a Service anbieten zu können. Der Code landet letztlich in Docker-Containern zur Ausführung (Abb. 2).

Manta von Joyent

Ein Sonderfall unter den Angeboten ist Joyents Manta. Zwar gehört es zu den älteren Angeboten, allerdings bietet es keine Trigger an außer dem manuellen Anstoßen der Funktionen, die hier Jobs heißen. Im Kern handelt es sich bei Manta um einen Objektspeicher. Anwender können die hochgeladenen Objekte direkt auf dem Server bearbeiten, ohne dafür zusätzliche Compute-Leistung erwerben oder einrichten zu müssen. Joyent stellt hierfür Kommandozeilenwerkzeuge zur Verfügung. *mput* lädt Dateien in den Speicher. In Kombination mit *mls* und *mjob* lassen sich die Uploads bearbeiten:

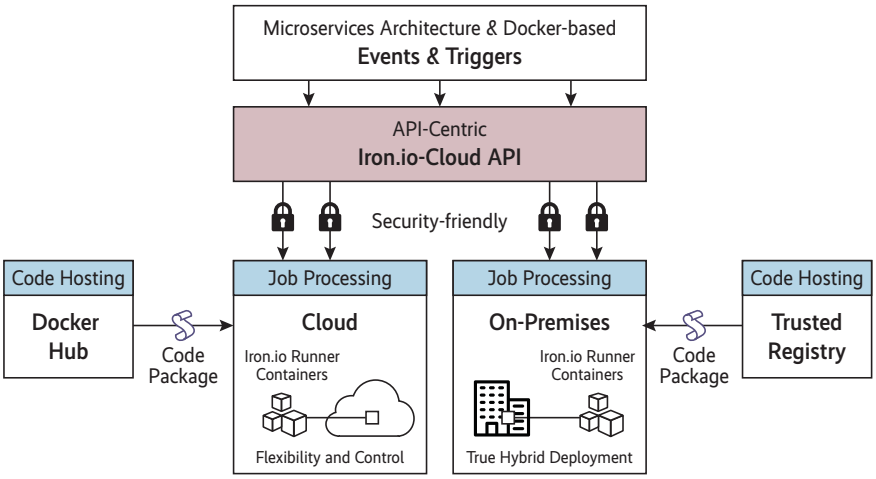
```
mfind /$MANTA_USER/public | ↵
                            mjob create -w -m ↵
                  'convert $MANTA_INPUT_FILE ↵
                      -resize 100x jpeg:- | ↵
        mpipe ${MANTA_INPUT_OBJECT%.*}-100x.jpg'
```

Eine Manta-Instanz läuft auf dem Open-Solaris-Klon SmartOS. Darauf setzt Joyents Cloud-Verwaltung Triton auf, die als Dienst wiederum Manta anbietet. Das lässt sich on Premises einrichten [2] oder aus Joyents Public Cloud beziehen. Die enge Verknüpfung von Speicher und Jobs legt als Anwendungsfall Stapelverarbeitung nahe. Datenverarbeitung lässt sich zum Beispiel realisieren, indem Anwender JSON-Dateien in den Speicher hochladen und dort mit den entsprechenden Tools darauf reagieren. Die Trigger müssen sie jedoch über externe Dienste einrichten, da sie kein Bestandteil von Manta sind.

Auf der anderen Seite ist Joyent relativ großzügig, welche Programme und Sprachen erlaubt sind. Die Jobs laufen in Solaris-Zonen und nutzen damit eine ausgereifte Technik, um sie vom restlichen System oder anderen Nutzern zu trennen. Für jeden Jobaufruf erstellt Manta per Copy-on-Write eine neue Zone aus einem Basisabbild. Alle Änderungen daran ver-

schwinden nach dem Löschen der Zone durch ein *zfs rollback*.

Prinzipiell könnten Administratoren ins Abbild beliebige Software einfügen. Vorinstalliert sind eine Reihe von Kommandozeilenwerkzeugen wie ImageMagick zur Bildbearbeitung. Codebeispiele finden sich vorwiegend in Node.js, dessen Entwicklung Joyent stark fördert. Zusätzlich versteht Manta Python, Java, Bash, PHP, Ruby, Perl und Go. Die Gnu Compiler Collection und gängige Datenbank-Clients gehören ebenfalls zur Standardausstattung. Individuelle Abläufe laden Nutzer als Manta-Assets hoch, um sie in der Cloud auszuführen.



Iron.io gehört neben Manta und Open Whisk zu den Diensten, die sowohl als Public-Cloud- als auch als On-Premises-Variante zur Verfügung stehen. Darüber hinaus erlaubt es den kombinierten Einsatz als Hybrid (Abb. 3).

hook.io

hook.io versteht sich als reines Webhook-Angebot, versteht also als Trigger nur HTTP-Anfragen. Ursprünglich ausschließlich für Node.js entwickelt, befindet sich das Produkt schon deutlich länger auf dem Markt als AWS Lambda. Inzwischen dürfte es die Plattform mit der längsten Liste kompatibler Sprachen sein.

Jeder Account ist gleichzeitig ein eigener Namespace, in dem neue Funktionen leben. Zugriff auf Funktionen erhält man über die URL http://hook.io/<user>/<function>. Als Funktionsattribute lassen sich GET- und POST-Parameter übergeben. Durch Erweitern der URL gelangen die Entwickler auf weitere Inhalte: http://hook.io/<user>/<function>/_src/ zeigt den Quelltext – jedem, der diese URL aufruft. Um das zu verhindern, lässt sich eine Funktion als privat einrichten, jedoch nur in der kostenpflichtigen Variante. Ähnlich gelangt man durch Erweitern des Pfads um */admin* zur Verwaltung der Funktion, sofern man der Besitzer ist. Möchte man die Funktion eines anderen Nutzers verwenden, geht dies über */_fork*.

Die Hooks verstehen auch binäre Datenströme und lassen sich somit in Kommandozeilen-Workflows einbinden:

```
cat image.png | curl -T -
                https://hook.io/user/modify_image
```

Zum Einbinden von Speicherdiensten haben alle Accounts Zugriff auf das integrierte virtuelle Dateisystem. hook.io stellt aber auch ein SDK zur Verfügung, mit dem sich externe Speicherdienste einbinden lassen. Zur Auswahl stehen SFTP/SSH, Amazon S3 sowie die Cloud-Speicher von Microsoft, Google und Rackspace.

Die kostenfreie Variante erlaubt 1000 monatliche Anfragen und erlegt den Funktionen ein Timeout von 10 Sekunden auf. In allen bezahlten Modellen ist es möglich, private Hooks anzulegen, zudem wächst mit dem Preis die Zahl der erlaubten monatlichen Anfragen. An der Spitze des Angebots steht die Enterprise-Variante, die bis zu 5000 Requests pro Sekunde abhandeln können soll und auf dedizierten Servern liegt.

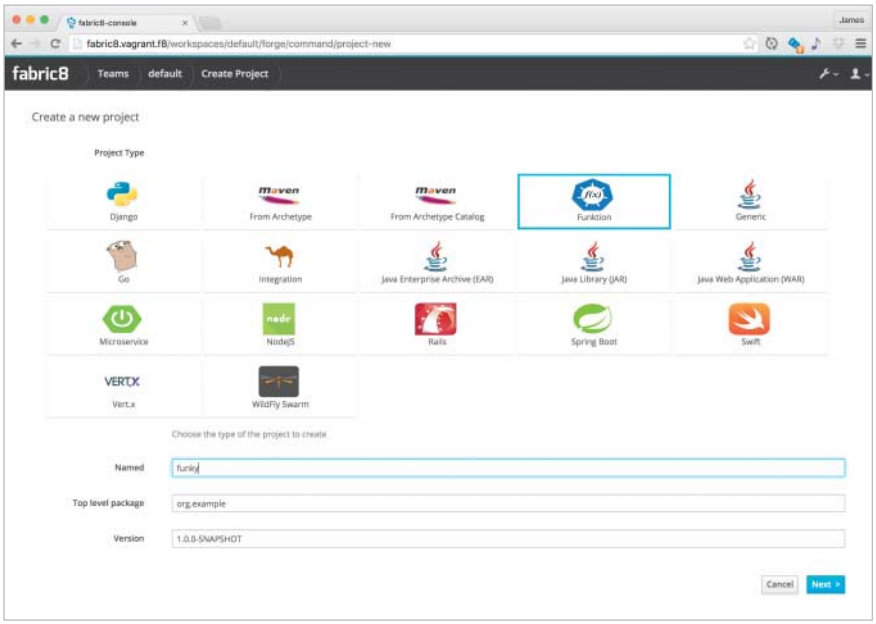
Iron.io

Mit der Kombination von IronMQ, IronWorker und IronCache bietet Iron.io

Functions as a Service aus der Public Cloud an. IronMQ stellt dabei die Message Queue, IronCache einen Key-Value-Store, die eigentliche Arbeit übernimmt IronWorker, der gegenwärtig mit Ruby, PHP, Python, Java, Node.js, Go und .NET umgehen kann. Das Angebot richtet sich allerdings eher an Eingeweihte, die Dokumentation enthält viele tote Links. Das Einrichten von Workern und Ausgaben funktioniert nur über die Kommandozeile und erfordert, dass man ein Image im Docker Hub ablegt oder Code bei GitHub einstellt.

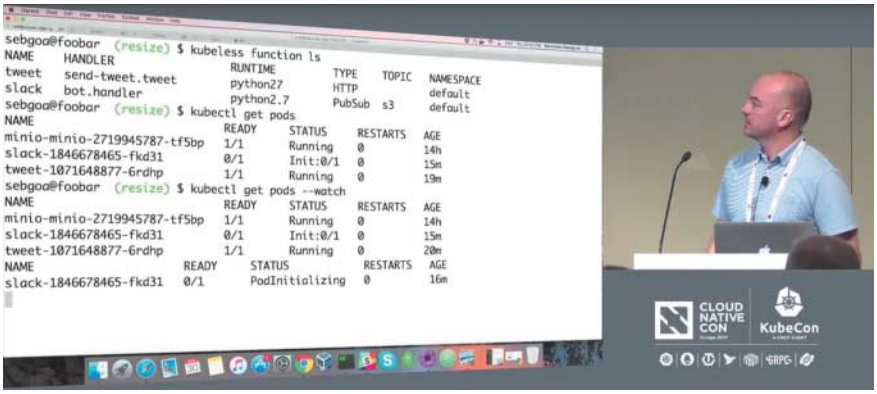
Ende 2016 hat Iron.io sein Open-Source-Project IronFunctions vorgestellt. Laut Roadmap sollte es seit März 2017 allgemein verfügbar sein, tatsächlich befindet sich das Projekt noch in der Betaphase. Die Software setzt zentral auf Docker und Container. Im Prinzip sind die Runtimes der Funktionen Docker-Images mit festgelegtem Entrypoint, der den jeweiligen Code interpretiert. Dadurch können Anwender individuelle Pakete zusammenstellen. Iron.io stellt im Docker Hub vorgefertigte Images für eine Reihe von Sprachen bereit. Davon, dass das Projekt noch nicht fertig ist, zeugen unter anderem die unvollständigen und fehlenden Beschreibungen der Muster-Runtimes.

Entwickler können bereits den IronFunctions-Server installieren. Der wird ebenfalls als Docker-Image bereitgestellt. Auch künftig können Nutzer den Server on Premises installieren. Mit Functions-UI steht ein Web-UI bereit. Die Picasso-API für OpenStack soll IronFunctions verwenden, um Functions as a Service innerhalb eines OpenStack-Clusters zu implementieren und in der Private Cloud eine skalierende Serverless-Infrastruktur bereitzustellen.



Quelle: Fabric8

Funktion existiert als eigenständiges Projekt, lässt sich aber auch in die Cloud-Entwicklungsplattform Fabric8 einbinden. Dort lässt es sich über das Web-GUI ansprechen (Abb. 4).



Quelle: Cloud Native Computing Foundation

Sebastien Goasguen präsentiert Kubeless auf der CloudNativeCon 2017 in Berlin. Der Dienst ist tief in Kubernetes eingebunden (Abb. 5).

Webtask.io

Eigentlich für Single Sign-on bekannt, hat Auth0 mit Webtask eine simple Plattform für Functions as a Service auf den Markt gebracht. Anhand der Beispiele in der Dokumentation stellt sich Webtask.io primär als Backend für Webapplikationen dar. Die Steuerung erfolgt über einen Node.js-Client, über den Nutzer sich auch anmelden können:

```
npm install wt-cli -g
```

Dieser Client erzeugt Funktionen im Backend:

```
wt create foo.js
```

Ohne weitere Angaben interpretiert Webtask.io den Code in der Datei *foo.js* als JavaScript. Es bietet Compiler für weitere Sprachen, zum Beispiel C#, TypeScript oder T-SQL, die allerdings immer zu JavaScript kompilieren. Anstelle eines Dateipfads lassen sich auch URLs angeben, etwa zu Dateien auf GitHub. Neben dem Aufruf über die HTTP-API beherrscht Webtask regelmäßig auszuführende Aufgaben *(wt cron)*.

Die Verwendung von Webtask erfordert einen Account bei Auth0. Damit lässt sich relativ einfach Authentifizierung über die Dienste von Auth0 integrieren. Diese Verbindung ist das hauptsächliche Verkaufsargument des Dienstes. Im Vergleich zu den anderen Angeboten schränkt Auth0 seine Kunden stark ein, was die Auswahl an Sprachen und Erweiterungen betrifft. Dafür erlegt das Unternehmen diesen keine mengenmäßigen Beschränkungen bei der Nutzung der Webtasks auf. Kostenpflichtige Zugänge erlauben mehr Optionen für die Authentifizierung, etwa die Verknüpfung mit sozialen Netzwerken oder einem Active Directory.

Fission

Fission von Platform9 ist fest mit Kubernetes verknüpft, läuft also auch nur darauf. Da sich die Orchestrierungsplattform aber inzwischen auf sehr unterschiedlicher Infrastruktur installieren lässt, schränkt das die Nutzbarkeit von Fission nicht ein.

Der Dienst beruht auf drei Säulen: Umgebungen *(environments)*, Funktionen *(functions)* und Routen *(routes)*. Mit Umgebungen bezeichnet Fission Laufzeitumgebungen. Zurzeit bietet die Plattform Environments für Python, Node.js, Go,

C# und PHP. Diese sind jedoch nicht von Haus aus aktiviert, sondern müssen über die Kommandozeile installiert werden, damit Funktionen anschließend darin abgelegt werden können:

```
$ fission env create --name nodejs --image
fission/node-env
$ fission function create --name hello --env
nodejs --code hello.js
```

fission function create kopiert den Code aus einer Datei in einen Container. Damit die Funktion arbeiten kann, müssen Anwender ihr per *fission route add* Routen zuweisen. Das können HTTP-Requests sein oder Kubernetes-Events. Über Requests lässt sich Fission mit den Diensten anderer Anbieter verknüpfen, die Verbindung mit Kubernetes lässt sich zum Beispiel dafür nutzen, die Cluster-Administration teilweise zu automatisieren.

Fission definiert die Umgebungen über Dockerfiles, sodass sich leicht Modifikationen vornehmen lassen. Nutzer können dadurch Erweiterungen oder spezielle benötigte Module mit einbinden.

Momentan befindet sich Fission noch im Alphastadium. Der Einsatzbereich dürfte wenig Überschneidungen mit den Angeboten der großen kommerziellen Anbieter besitzen. Mehrere Mandanten sind bisher nicht möglich und auch in der näheren Zukunft nicht angedacht. Dadurch schränkt sich der Anwenderbereich ein auf Administratoren und Rechenzentrumsbetreiber, die nur Funktionen aus vertrauenswürdigen Quellen einsetzen.

Funktion.io

Ebenfalls auf Kubernetes (und auf OpenShift) läuft das Function-as-a-Service-Framework Funktion.io. Das wiederum ist eng verknüpft mit der Cloud-Entwicklungsplattform Fabric8. Die Entwicklung übernimmt zu großen Teilen Red Hat. Funktion.io installiert sich automatisiert als Kubernetes-Deployment in Kubernetes und OpenShift und lässt sich mit Fabric8 kombinieren.

Die Funktionen selbst bekommen eine Runtime zugewiesen. Node.js ist die einzige, die sich über *funktion install runtime* installieren lässt. Die Dokumentation schweigt sich darüber aus, wie Anwender den Sprachumfang erweitern können. Überhaupt gibt das Onlinehandbuch nur sehr vagen Einblick in das Framework. Der einfachste Aufruf erfolgt über einen HTTP-Endpunkt. Um auf Events zu reagieren, verwendet Funktion.io Apache Camel. Somit stehen alle dort definierten Endpunkte zur Verfügung, die Funktion.io aufrufen können. Das geschieht über Flows, die zwischen Ereignissen und Aufrufen mitteln.

Die Objekte sind durchgehend als Kubernetes' *ConfigMaps* verpackt, das Programm *funktion* übersetzt also primär zwischen der Welt der Microservices und Kubernetes beziehungsweise OpenShift. Wer Funktion.io auf Fabric8 installiert, erhält dort ein grafisches Interface, um die Funktionen zu verwalten (siehe Abbildung 4).

Kubeless

Ein weiteres Framework für Serverless unter Kubernetes ist Kubeless. Von den anderen Kandidaten unterscheidet es sich dadurch, dass es als Third Party Resource in Kubernetes implementiert ist. Dadurch werden die Funktionen zu eigenständigen Instanzen und lassen sich etwa mit *kubectl get functions* abfragen. Als Trigger stehen HTTP-Endpunkte oder PubSub-Topics zur Verfügung. Letztere sammelt ein Kafka-Pod, der fester Bestandteil einer Kubeless-Installation ist.

Mit dem Kommando *kubeless* lassen sich Funktionen definieren:

```
kubeless function deploy test \
    --runtime python27 --handler test.foobar \
        --from-file test.py --trigger-topic \
                                <<topic_name>
```

Statt *--trigger-topic* versteht der Befehl auch *--trigger-http* und erzeugt einen passenden Endpunkt. Auch das Topic erzeugt man mit *kubeless*: *kubeless create topic <topic_name>*. Neben Kafka läuft der Kubeless-Controller, der die Runtimes bereitstellt. Derzeit versteht Kubeless Node.js und Python.

Fazit

Unter dem Begriff „Serverless" wird eine große Bandbreite von Diensten subsumiert, die teilweise recht wenig gemeinsam haben. Das klassische Beispiel, in dem eine solche Plattform zum Einsatz kommt, um reihenweise Bilder umzuskalieren, funktioniert bei allen vorgestellten Anbietern. Die Dienste von Amazon, Microsoft und Google bieten Kunden den Vorteil, auf eine umfangreiche Landschaft an Zusatzdiensten und hilfreiche Ressourcen zurückgreifen zu können. Ähnlich gilt das für Open Whisk, das immer noch stark in IBM Bluemix eingebunden ist und etwa eine Schnittstelle zu den Watson-Diensten besitzt. Entwickler machen sich dann allerdings davon abhängig. Diesem Vendor-Lock-in entgeht man, wenn man auf diese Vorzüge ver-

Marktübersicht Plattformen für Serverless Services

	AWS Lambda	Azure Functions	Cloud Functions	Open Whisk	Manta	hook.io	Iron Functions & IronWorker	Webtask.io	Fission	Funktion	Kubeless
Hersteller	Amazon	Microsoft	Google	IBM / Apache Software Foundation	Joyent	hook.io	Iron.IO	Auth0	Platform 9	Red Hat	Bitnami
Website	https://aws.amazon.com/de/lambda/	https://azure.microsoft.com/de-de/services/functions/	https://cloud.google.com/functions/	http://openwhisk.org/	https://github.com/joyent/manta	https://hook.io	https://www.iron.io/	https://webtask.io/	https://platform9.com/fission	https://funktion.fabric8.io/	https://github.com/bitnami/kubeless
Cloud	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
on Premises	–	geplant	–	✓	✓	–	✓	–	✓	✓	✓
Lizenz				Apache 2.0	Mozilla Public License Version 2.0		Apache 2.0		Apache 2.0	Apache 2.0	Apache 2.0
Voraussetzung				Docker	SmartOS		Docker > 1.12		Kubernetes	Kubernetes oder OpenShift	Kubernetes
Sprachen											
Node.js / JavaScript	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Java	✓			✓	✓		✓				
C#	✓	✓									
F#		✓									
Python	✓	✓		✓	✓	✓	✓		✓		✓
Swift				✓							
Bash		✓			✓	✓	✓				
Batch		✓									
Coffee-Script						✓					
Lua						✓					
PHP		✓			✓	✓	✓				
Ruby					✓	✓	✓				
Scheme						✓					
SmallTalk						✓					
Tcl						✓					
Perl					✓		✓				
Go					✓						
PowerShell		✓									
Mono							✓				
Besonderheiten					zusätzlich: GCC, Bild- und Videobearbeitung, Datenbank-Clients		individuelle Dockerfiles mit Entrypoint	erlaubt das Einbinden von Compilern, die Code in JavaScript übersetzen			

zichtet. Dadurch vergrößert sich auch die Auswahl an Diensten.

Offensichtlich lassen sich Serverless Services gut im Continuous-Integration-Umfeld einsetzen, besonders für JavaScript. Sonst ist nur Python ähnlich universell verfügbar. Wer sich nicht auf eine Sprache festlegen will, sollte vor der Wahl des Anbieters überlegen, welche Runtimes wichtig sind.

Völlig in eine andere Richtung geht der Ansatz von Fission, das erlaubt, DevOps weitgehend zu ignorieren und sich stattdessen komplett im Ops-Bereich zu bewegen und die Administration eines Kubernetes-Clusters um automatische Helfer zu erweitern. Ähnlich lassen sich alle Kubernetes-Aufsätze für Serverless Services verwenden, aber auch die Dienste der großen Anbieter verstehen deren Cloud-eigenes Monitoring-System. Amazon etwa kann eine Funktion auslösen, wenn CloudWatch einen Fehler meldet.

Eine weitere Einsatzmöglichkeit bieten die Functions as a Service mit der Verarbeitung von Daten aus vernetzten Geräten (IoT) oder großer Datenmengen, die unregelmäßig eintrudeln.

Der Reiz von Serverless Services liegt in der hohen Skalierbarkeit, die besonders durch die sekundengenaue Abrechnung gegeben ist. Wer davon ausgehen kann, Rechenleistung über einen längeren Zeitraum oder sogar kontinuierlich zu brauchen, benötigt eine andere Art von Cloud, da dann die Kosten von VM-Instanzen niedriger liegen. (jab)

Literatur

[1] Constantin Söldner; Serverless Programming; Tröpfchenweise; Abkehr vom monolithischen Entwickeln; *iX* 8/2016, S. 78

[2] Töns Büker; Serverless Computing; Fleißiger Speicherdienst; Objekte speichern und bearbeiten mit Manta; *iX* 4/2017, S. 118

Alle Links: www.ix.de/ix1706034