Ein unvollständiger Überblick

Microservice-Infrastruktur

Michael Vitz

Die Cloud und darauf aufbauend Microservices haben in den letzten Jahren die Softwareentwicklung stark beeinflusst. Vor allem im Bereich der Infrastruktur tut sich hier auch heute noch einiges. Dabei wird man den Eindruck nicht los, dass jede Woche ein neues Produkt veröffentlicht wird. In diesem Artikel versuche ich, Ihnen durch diesen Dschungel zu helfen.

Erst brachen die Cloud und anschließend Microservices über uns herein. Beides sorgt nach wie vor für eine Menge Bewegung im Markt. Dabei tauchen viele Begriffe, Abkürzungen und Akronyme auf, die es uns schwer machen, den Überblick zu behalten.

Dieser Artikel stellt Ihnen vier Kategorien von Infrastrukturkomponenten vor, die einem beim Betrieb einer in Microservices aufgeteilten Anwendung mit hoher Wahrscheinlichkeit begegnen: Plattform, Edge-Services, Service-Registry und Konfiguration (s. Abb. 1).

Zu jeder Kategorie gehört neben einer Erklärung ihrer Funktion und dem Aufzeigen, wieso diese Funktion für eine Microservice-Architektur relevant ist, auch eine Tabelle von Produkten, die für diese Funktion eingesetzt werden können.

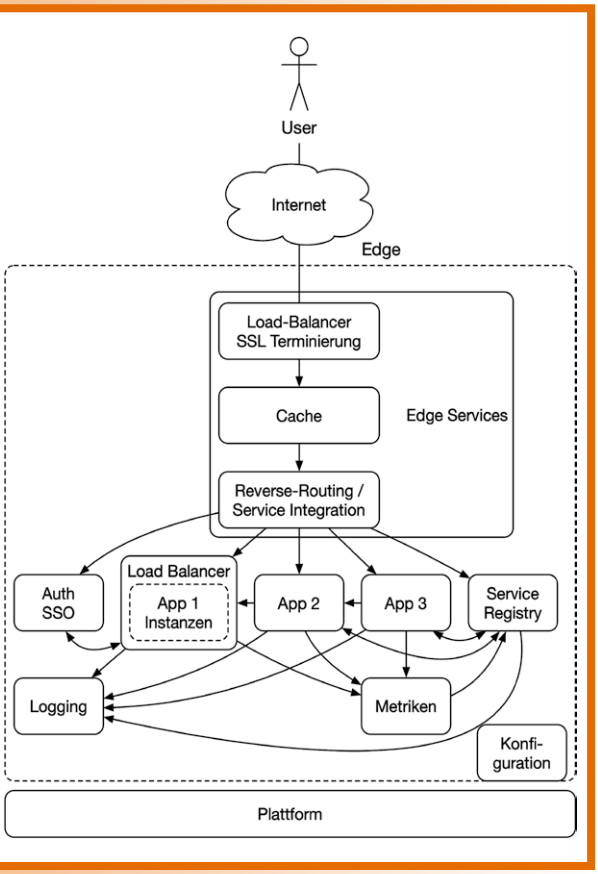

Abb. 1: Microservice-Architektur Blueprint

Plattform

Da eine Anwendung heute häufig aus mehreren verteilten Services besteht, haben sich die Anforderungen an die Plattform geändert, auf der die Services laufen sollen. Kam man vor ein paar Jahren noch sehr gut damit aus, einen neuen Server beim Betrieb per Formular zu bestellen, wird heute erwartet, dass man sich per Selbstbedienung innerhalb weniger Minuten eine neue Laufzeitumgebung erstellen kann.

Um diesen Anforderungen zu genügen, nutzt man am besten eine fertige Lösung, anstatt das Rad neu zu erfinden. Grundsätzlich lassen sich die Lösungen in vier Kategorien aufteilen:

- Infrastruktur as a Service (*IaaS*) stellt dem Nutzer einfach und schnell neue virtuelle Umgebungen zur Verfügung. Diese Produkte stellen somit die geringste Abstraktion von „realer" Hardware dar.
- Ähnliche Ziele hat Container as a Service (*CaaS*), das die neuste der vier Plattformkategorien darstellt. Durch die Nutzung von Containern lassen sich hier im Gegensatz zu IaaS häufig die vorhandenen Ressourcen besser nutzen, da der Overhead eines Containers zu einer virtuellen Maschine geringer ist.
- Plattform as a Service (*PaaS*) bietet eine höhere Abstraktion an. Der Nutzer muss sich nicht mehr um Details, wie das konkrete Betriebssystem, kümmern, sondern wählt eine von der Plattform angebotene Laufzeitumgebung. Dies schränkt die Freiheiten für den Entwickler natürlich ein, nimmt ihm jedoch auch viele unangenehme Dinge ab. So muss man sich zum Beispiel um Sicherheitsupdates des Betriebssystems keine Gedanken mehr machen, da dies Aufgabe der Plattform ist. Zudem bieten die meisten PaaS-Lösungen Infrastruktur-Services an. So lässt sich zum Beispiel eine Datenbank mit wenigen Klicks aufsetzen und in die eigene Anwendung integrieren.
- Software as a Service (*SaaS*) als vierte und letzte Kategorie ist für die Entwicklung meistens von wenig Interesse und wird deswegen hier nicht weiter erwähnt.

Tabelle 1 gibt einen Überblick über aktuell auf dem Markt verfügbare Lösungen in der Kategorie, in der diese sich primär befinden.

Plattformlösung	Kategorie
CloudFoundry [CloudFoundry]	PaaS
CloudStack [CloudStack]	IaaS
Deis [Deis]	PaaS
Kubernetes [Kubernetes]	CaaS
Mesos [Mesos]	CaaS
OpenShift [OpenShift]	PaaS
OpenStack [OpenStack]	IaaS
Proxmox [Proxmox]	IaaS
Rancher [Rancher]	CaaS
Swarm [DockerSwarm]	CaaS

Tabelle 1: Produkte für die eigene Microservice-Infrastruktur

Kann man auf eine Public Cloud wie AWS [AWS], Google [GoogleCloud] oder Azure [Azure] setzen, so enthält diese in der Regel Dienste aus allen vier Kategorien. Solange Portabili-

tät zwischen den verschiedenen Clouds kein primäres Qualitätsziel ist, bietet es sich dabei an, die spezifischen Services des Anbieters zu nutzen.

Edge-Services

Auch wenn in einer Microservice-Architektur eine Anwendung in mehrere Services aufgeteilt wird, soll dies für den Nutzer unsichtbar sein. Deshalb ergibt sich an der Grenze der eigenen Infrastruktur (= der „Edge“) der Bedarf für Infrastrukturkomponenten.

Edge-Services decken dabei eine breite Palette von Funktionen ab: Load Balancing, Caching, Reverse Routing, SSL-Terminierung und gegebenenfalls Serviceintegration. Daher lohnt sich hier zumeist auch der Einsatz unterschiedlicher Produkte.

Die erste Aufgabe der Edge-Services liegt darin, für die eingehende Verbindung SSL zu terminieren. Je früher diese Terminierung stattfindet, desto geringer sind die Latenzen für den Nutzer. Zudem muss die SSL-Terminierung vor dem Einsatz eines Caches stattfinden, da dieser sonst nur die verschlüsselte Anfrage als Key nutzen könnte.

Die nächste Aufgabe besteht darin zu verhindern, dass der Kontaktpunkt zum Nutzer ein Single Point of Failure wird. Eine mögliche Lösung ist der Einsatz von DNS. Hierzu hinterlegt man mehrere IPs für eine Domain. Aufgrund der langen erlaubten Cache-Zeiten von DNS-Einträgen ist dies jedoch nicht sonderlich flexibel. Eine bessere Alternative bietet somit der Einsatz von Hardware oder Software Load Balancern. Diese nutzen in der Regel das ARP-Protokoll, um beim Ausfall einer Instanz den Traffic mit einer anderen Instanz zu übernehmen.

Da nicht jeder Request von einem Nutzer bis ins Backend gelangen muss, bietet es sich zudem an, Caching in den Edge-Services zu betreiben. Besonders für statische Assets wie Bilder, CSS, Web-Fonts oder JavaScript ist dies mit wenig Aufwand möglich. Aber auch Backend-Anfragen, die mit den passenden Caching-Headern ausgestattet wurden, können an dieser Stelle problemlos gecacht werden.

Die letzte Aufgabe der Edge-Services besteht darin, die Aufrufe der Nutzer zur richtigen Applikation zu routen. Zusätzlich erlauben es Techniken wie Edge Side [ESI] oder Server Side Includes [SSI], per Include-Direktiven Referenzen aufzulösen, sodass Clients zusammengesetzte Antworten erhalten können. ESI hat hierbei den Vorteil, dass die Referenzen von unterschiedlichsten Services geliefert werden können und dass das Caching dieser Fragmente bereits mit im Standard spezifiziert ist.

Tabelle 2 listet diverse Produkte auf, die für die verschiedenen Aspekte von Edge-Services eingesetzt werden können.

Service-Registry

Die nächste Herausforderung in einem verteilten System liegt darin, dass sich die Serviceinstanzen untereinander im Netzwerk finden müssen. Genau hierin liegt der Anwendungszweck von Service-Registrys.

Das Registrieren der Serviceinstanzen geschieht eigentlich immer über eine proprietäre Programmierschnittstelle der Registry. Je nach Wahl der Plattform lassen sich an dieser Stelle fertige Produkte einsetzen, um die Registrierung zu vereinfachen.

Um die Registry abzufragen, gibt es verschiedene Wege, die vom eingesetzten Produkt abhängen. Der erste Weg nutzt DNS zur Abfrage. Da dies im eigenen Netz geschieht, sind hier die Auswirkungen durch Caching absehbar und kontrollierbar. Alternativ bieten die meisten Service-Registrys eine HTTP-Programmierschnittstelle an. Die dritte Möglichkeit besteht darin, neben jedem Service einen weiteren Prozess zu starten, der von der Service-Registry über Änderungen informiert wird und anschließend ein benutzerdefiniertes Kommando ausführt. Hierbei kann man zum Beispiel eine neue Konfigurationsdatei erzeugen und anschließend den Service dazu bringen, die neue Konfiguration zu verwenden. Natürlich können hierbei Platzhalter durch Werte aus der Service-Registry ersetzt werden. Tabelle 3 listet einige Kandidaten für eine Service-Registry auf.

Produkt	Abfrageschnittstellen
Consul [Consul]	DNS, HTTP API, Templates [ConsulTemplate]
Eureka [Eureka]	HTTP API
SkyDNS [SkyDNS]	DNS

Tabelle 3: Service-Registrys

Konfiguration

Die nächste Aufgabe, für die man Infrastruktur einsetzen kann, ist die Konfiguration. Hierfür sollten wir zunächst klären, welche Dinge man in einer Microservice-Architektur konfigurieren möchte.

Produkt	Features
Apache HTTPD [HTTPD]	Caching, Load Balancing, Reverse Routing, SSI, SSL-Terminierung
Apache Traffic Server [TrafficServer]	Caching, Load Balancing
HA Proxy [HAProxy]	Load Balancing, SSL-Terminierung
NGINX [NGINX]	Caching, Load Balancing, Reverse Routing, SSI, SSL-Terminierung
Træfik [Traefik]	Load Balancing, Reverse Routing, SSL-Terminierung
Varnish [Varnish]	Caching, ESI, SSL-Terminierung (mit Hitch [Hitch])

Tabelle 2: Edge-Service-Produkte

So gut wie jeder Service benötigt Credentials. Der klassische Anwendungsfall hierzu sind technische Nutzer, die benötigt werden, um auf andere Systeme, wie die Datenbank, zuzugreifen. Aber auch der Zugriff auf andere Services ist häufig mit einem technischen Nutzer gesichert. Da Credentials besonders schützenswert sind, bietet es sich für diesen Aspekt an, eine genau hierauf spezialisierte Lösung zu nutzen.

Neben Credentials gibt es innerhalb eines Service technische Parameter, die man zentral konfigurieren möchte. Hierzu gehören zum Beispiel die Größe von Thread- oder Connection-Pools oder die Anzahl an Retries für Jobs.

Im einfachen Fall sind diese Parameter über die gesamte Laufzeit einer Serviceinstanz statisch. Dann braucht man keine zentrale Infrastrukturkomponente, sondern kann die Konfiguration beim Deployment eines Service mitgeben. Dies kann über Umgebungsvariablen oder klassisch über Konfigurationsdateien geschehen.

Sollen die Parameter zur Laufzeit dynamisch veränderbar bleiben, gibt es zwei Möglichkeiten. Entweder man wandelt diese in statische Parameter um, indem man bei einer Änderung die betroffenen Services mit der neuen Konfiguration startet. Dies bietet den Vorteil, dass man keine weitere Komponente benötigt und der Service weniger komplex bleibt. Müssen die Parameter tatsächlich zur Laufzeit geändert werden, bietet sich der Einsatz einer Datenbank an, die sich auf der CP-Achse von CAP (siehe Kasten „CAP-Theorem") befindet. Dabei reicht es natürlich nicht, nur die neuen Parameter zu laden, sondern die gesamte Anwendung muss darauf fehlerfrei reagieren.

Neben den technischen Parametern zählen auch fachliche Parameter zur Konfiguration. Diese gehören jedoch meiner Meinung nach idealerweise direkt in den Service, der diese nutzt. Sollte dies nicht möglich sein oder Sie anderer Meinung sein, so lassen sich diese natürlich auch mit in einem System zur zentralen Konfiguration verwalten. Einen Überblick über Systeme, die für zentrale Konfiguration nutzbar sind, bietet Tabelle 4.

Produkt	Art von Konfiguration
Consul	Allgemein
etcd [etcd]	Allgemein
Keywhiz [Keywhiz]	Credentials
Vault [Vault]	Credentials
Zookeeper [Zookeeper]	Allgemein

Tabelle 4: Systeme zur Konfiguration

Fazit

Ich hoffe, der Artikel hat Ihnen einen Überblick über einige der in einer Microservice-Architektur oft genutzten Infrastrukturkomponenten gegeben. Aufgrund des zur Verfügung stehenden Platzes fehlen noch weitere Kategorien wie Logging, Metriken oder Authentifizierung. Auch lassen sich für die im Artikel genannten Kategorien mit Sicherheit noch weitere mögliche Produkte finden.

Deswegen bitte ich Sie darum, bevor Sie nun losziehen und eines der hier vorgestellten Dinge oder Produkte in Ihrer Anwendung einführen, zu überlegen, welches Problem Sie gerade lösen müssen, und zu prüfen, ob das Muster oder Produkt dieses Problem wirklich löst.

Links

[AWS] https://aws.amazon.com
[Azure] https://azure.microsoft.com
[CAP] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.1495&rep=rep1&type=pdf
[CloudFoundry] https://www.cloudfoundry.org
[CloudStack] https://cloudstack.apache.org
[Consul] https://www.consul.io
[ConsulTemplate] https://github.com/hashicorp/consul-template
[Deis] https://deis.com
[DockerSwarm] https://www.docker.com/products/docker-swarm
[ESI] https://www.w3.org/TR/esi-lang
[etcd] https://github.com/coreos/etcd
[Eureka] https://github.com/Netflix/eureka
[GoogleCloud] https://cloud.google.com
[HAProxy] http://www.haproxy.org
[Hitch] https://www.hitch-tls.org
[HTTPD] https://httpd.apache.org
[Keywhiz] https://square.github.io/keywhiz/
[Kubernetes] http://kubernetes.io
[Mesos] http://mesos.apache.org
[NGINX] https://www.nginx.com
[OpenShift] https://www.openshift.org
[OpenStack] https://www.openstack.org
[Proxmox] http://proxmox.org
[Rancher] http://rancher.com/rancher/
[SkyDNS] https://github.com/skynetservices/skydns
[SSI] http://www.yourhtmlsource.com/sitemanagement/includes.html
[Traefik] https://traefik.io
[TrafficServer] http://trafficserver.apache.org
[Varnish] https://www.varnish-cache.org
[Vault] https://www.vaultproject.io
[Zookeeper] https://zookeeper.apache.org

CAP-Theorem

Das CAP-Theorem [CAP] besagt, dass es in einem verteiltem System nicht möglich ist, gleichzeitig die drei Eigenschaften Konsistenz (C), Verfügbarkeit (A) und Partitionstoleranz (P) vollständig zu erfüllen. Man muss bei verteilten Systemen zwischen Konsistenz und Verfügbarkeit abwägen und graduell entscheiden.

Ein System ist hierbei konsistent, wenn eine lesende Operation garantiert immer den zuletzt geschriebenen Wert zurückgibt; verfügbar, wenn selbst bei Ausfall eines Knotens das System weiterhin antwortet, und partitionstolerant, wenn das System auch dann funktioniert, wenn Sie Teile des Systems nicht mehr im Netzwerk erreichen können.

Michael Vitz ist Consultant bei innoQ und verfügt über mehrjährige Erfahrung in der Entwicklung und im Betrieb von JVM-basierten Systemen. Zurzeit beschäftigt er sich vor allem mit den Themen Microservices, Cloud-Architekturen, DevOps, Spring-Framework sowie Clojure.
E-Mail: michael.vitz@innoq.com