OpenStack: Interner Aufbau

Eine Seminar-Arbeit bei Prof. Dr. -Ing. Dr. rer. nat. habil. Harald Richter

Name	Christian Rebischke
Studiengang	Informatik
Matrikelnummer	432108
Datum	07.12.2017

Inhaltsverzeichnis

Vorwort

Keine andere Technologie wirbelt im Augenblick die Industrie so sehr auf wie Cloud-Computing. Cloud-Computing scheint so etwas wie die Basis zum Erfolg zu sein, der Antriebsmotor der Digitalisierung. Durch Cloud-Computing ist es möglich kostensparsam und schnell (on-demand) Ressourcen zu buchen und zu nutzen, die ohne Cloud-Computing nur durch teure Selbstbeschaffung zugänglich wäre. Doch was ist Cloud-Computing eigentlich? Das „National Institute of Standards and Technology“, kurz *NIST*, schreibt dem Cloud-Computing folgende Charakterisika zu [1]:

- **On-demand self-service**: Ein Kunde kann möglichst ohne viel Aufwand Ressourcen in der Cloud buchen ohne dabei auf menschliche Interaktion angewiesen zu sein. Als Ressourcen zählen Systemressourcen wie Rechenzeit, Speicher oder Netzwerkkapazität.
- **Broad network access**: Die Grundlage für den Zugang zur Cloud bietet das Internet. Zum Zugriff können alle gängigen Instrumente verwendet werden: Smartphone, Desktop-PC, Laptop. Der Zugang findet meistens über den Browser statt, ist aber nicht verpflichtend. Es gibt auch Clouds, welche eine *API* zur Verfügung stellen. Durch diese *API* lässt sich auch der Zugriff durch andere Software realisieren. Beispielsweise über die Kommandozeile.
- **Resource pooling**: Ressourcen werden zusammengefasst zu einem großen Pool. Diese Ressourcen-Pools können dann unterteilt und an mehrere Kunden vermietet werden. Dabei ist der genaue Standort der Ressourcen für den Kunden prinzipiell irrelevant, obwohl einige Cloud-Anbieter, allen voran Amazon AWS, dem Kunden die Freiheit gibt den Standort selbst festzulegen.
- **Rapid elasticity**: Eine Cloud ist jederzeit und völlig automatisch skalierbar. Dies ermöglicht es innerhalb von Sekunden auf Server-Belastung zu reagieren und Ressourcen nachzubeordern. Aber auch nicht mehr benötigte Ressourcen wieder freizugeben.
- **Measured service**: Die Cloud überprüft und managed die Ressourcen automatisch. So werden nicht mehr benötigte Ressourcen freigegeben und Ressourcen die öfters benötigt werden reserviert. Der Ressourcenbedarf wird also überwacht, kontrolliert und bei Bedarf angepasst. Dies schafft Transparenz für den Cloud-Anbieter als auch für den Kunden.

Beispiele für solche Cloud-Computing-Plattformen sind beispielsweise OpenStack, OpenNebula, die Amazon AWS Cloud, die Microsoft Azura Cloud, der OpenStack *Fork* SysEleven Stack von der Berliner Firma SysEleven und die Google Cloud Plattform vom Software-Giganten Google beziehungsweise seinem Mutterkonzern Alphabet.

Der interne Aufbau von OpenStack

Einführung

Als Beispiel für den internen Aufbau einer Cloud dient in dieser Arbeit die Software OpenStack. OpenStacks Entwicklungsgeschichte reicht bis in das Jahr 2010 zurück.[2] Die Open-Source-Software wurde ursprünglich vom *Web-Hoster* Rackspace in Zusammenarbeit mit der *NASA* und der Firma Anso Labs ins Leben gerufen.[3] Die Firma Anso Labs wurde im Jahr 2011 von der Firma Rackspace übernommen.[4] Seitdem hat sich ein OpenStack-Konsortium gebildet, welches die weitere Entwicklung von OpenStack überwacht und steuert. Dieses Konsortium besteht aus mehreren Vertretern der Wirtschaft. Darunter auch deutsche Firmen, wie beispielsweise die Telekom, welche selbst einen Ableger von OpenStack einsetzt (Open Telekom Cloud).[5] Die OpenStack-Entwickler haben sich gemeinsam zum Ziel gesetzt eine „allgegenwärtige Open-Source-Cloud-Computing-Plattform zu bauen, welche einfach zu benutzen ist, simpel zu implementieren ist, gut zu anderen Bausteinen passt, frei skalierbar ist und alle Anforderungen erfüllt, die Endverbraucher und Administratoren an eine Public und Private Cloud stellen.“[6] Die Begriffe, auch bekannt als *Deployment Models*, Public und Private Cloud wurden vom *NIST* eindeutig definiert:[1]

- **Public cloud**: Die Cloud ist der breiten Öffentlichkeit frei verfügbar. Der Cloud-Betreiber kann eine Firma oder eine andere Institution sein, wie beispielsweise eine Universität.
- **Private cloud**: Die Cloud ist beschränkt auf eine Firma und deren internen Abteilungen.

Anmerkung: Das *NIST* definiert noch zwei weitere *Deployment Models* darunter fallen die Community Cloud und die Hybrid Cloud. Eine Community Cloud ist eine Cloud für mehrere Organisationen. Der Cloud-Anbieter ist dabei nicht zwingend eine der Organisationen. Auch ein unabhängiger Anbieter oder Sponsor ist denkbar. Die Hybrid Cloud ist ein Mix aus mehreren *Deployment Models*, beispielsweise wenn eine Firma eine Cloud sowohl intern benutzt als auch extern Kunden anbietet.

Technische Details

Infrastructure as a Service (IaaS)

Die OpenStack-Plattform folgt strikt dem *Service Model* des „Infrastructure as a Service“, kurz *IaaS.* Das *NIST* definiert *IaaS* wie folgt[1]:

- **Infrastructure as a Service**: Der Kunde hat die Möglichkeit beliebig viel Speicher, Netzwerkkapazität, Rechenzeit und andere fundamentale Ressourcen zu nutzen. Außerdem ist es ihm frei gestellt beliebige Software-Applikationen und Betriebssysteme auf diesen Netzwerkknoten zu installieren und host-basierte Firewall-Einstellungen zu ändern. Der Kunde hat allerdings keinen Zugriff auf cloud-interne Infrastruktur.

Diese Defintion als technische Basis nehmend, folgt daraus eine abstrahierte Darstellung der technischen Komponenten einer heutigen Cloud:

- **Apps**: Betriebssysteme und deren Software-Applikationen. Meistens Software-Applikationen die darauf ausgelegt sind dynamisch skalieren zu können. Zum Beispiel *OpenMPI* ein Framework um verteilt und effizient zu Rechnen.
- **Cloud**: Die Cloud an sich mit der Management-Oberfläche und internen Komponenten.
- **Netzwerkknoten**: Virtuelle Maschinen und Container mit zugewiesener Rechenzeit in Form von *CPU* und *RAM.*
- **Netzwerk**: hochverfügbare Netzwerk-Ressourcen
- **Storage**: hochverfügbare Speicher-Ressourcen.

Anmerkung: Das Institute of Electrical and Electronics Engineers (*IEEE*) definiert den Begriff der Hochverfügbarkeit (englisch: High Availability) wie folgt:

„High Availability (HA for short) refers to the availability of resources in a computer system, in the wake of component failures in the system.“[7] (deutsch: „Hochverfügbarkeit beschreibt die Erreichbarkeit von Computer-Systemen, im Falle des Ausfalls einer Komponente.“)

Die minimale Cloud

Um der Rolle als „Infrastructure as a Service“ gerecht zu werden ist OpenStack aus Modulen aufgebaut. Jedem dieser Module kommt genau eine Aufgabe zu. Ganz nach der *UNIX*-Philosophie von Douglas McIlroy[8]: „Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new ‚features‘“ (deutsch: „Schreibe ein Programm welches eine Sache gut macht. Für eine neue Aufgabe ist es besser ein neues Programm zu schreiben als dem alten Programm neue Funktionen hinzuzufügen“).

Die nachfolgende Grafik verdeutlicht die benötigten Komponenten um eine minimale Cloud mit OpenStack zu betreiben:

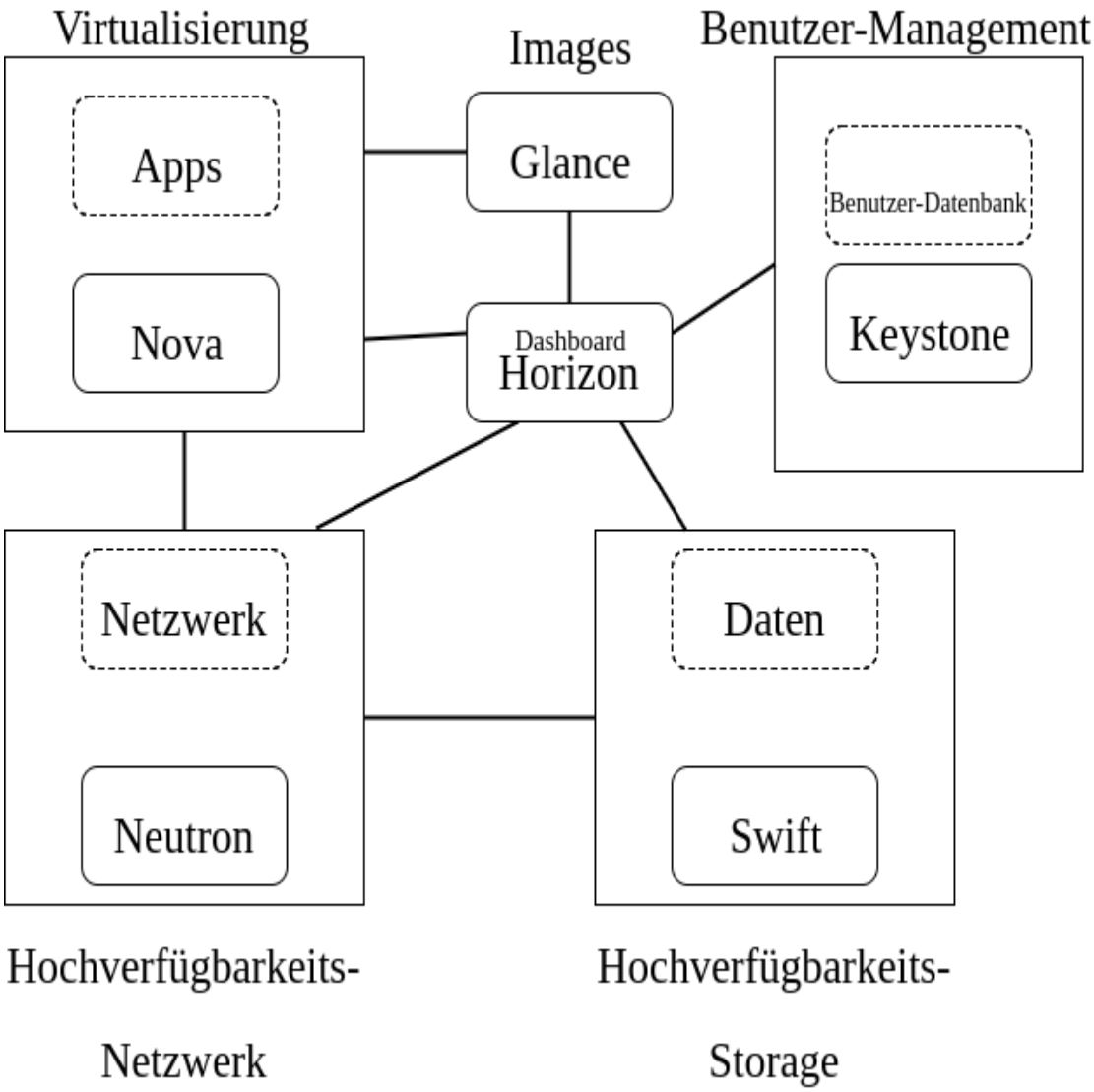

Abbildung 1: Eine minimale Cloud mit OpenStack

Horizon

Einer der Kernbestandteile der minimalen Cloud mit Openstack ist die Komponente *Horizon.* *Horizon* bietet ein Dashboard an um die übrigen Komponenten aus dem Browser heraus zu steuern. Dabei kommen *HTTP* auf Port 80 und *HTTPS* auf Port 443 zum Einsatz. Es basiert auf folgenden Programmier- und Auszeichnungssprachen[9]:

- 64.1% *Python*
- 27.4% *Javascript*
- 6.9% *HTML*
- 1.6% *CSS*

Aus der Aufteilung der Programmier- und Auszeichnungssprachen sowie aus der Struktur des Projekts wird ersichtlich, dass *Horizon* auf dem *Python*-Web-Framework *Django* basiert. So finden sich diverse Querverweise auf das *Django* Framework, wie beispielsweise einige Konfigurationsdateien[10]:

- **manage.py**: Dies ist ein *Python*-Script zur Verwaltung des *Django*-Projekts. (Installation, Ausführen diverser Test-Fälle, Migration der Datenbank, überprüfen des Systemzustands)
- **settings.py**: Einstellungen für das Projekt. (Hostname, Datenbank-Backend)
- **urls.py**: Die unterschiedlichen *URL*s des Projekts (Zum Beispiel der Administrator-Bereich)

Desweiteren wird aus dem Projekt ersichtlich, dass das *Horizon* Dashboard eine *REST-API* anbietet. Über diese *REST-API* kommunizieren dann die restlichen Komponenten mit dem *Horizon* Dashboard. Als Script-Sprache kann *Python* verwendet werden um auf diese *API* zuzugreifen[11]. Über diese *API* können dann Informationen abgerufen werden, Aktionen ausgeführt werden die normalerweise ein Benutzer am Rechner tun würde oder auch ein völlig eigenes Dashboard implementiert werden.

Das Dashboard selbst kann via Anpassungen des *CSS* an das *Corporate Design* der eigenen Firma angepasst werden.

Nova

Nova ist das Herz von OpenStack. Es spricht mit mehreren *Hypervisoren* und hilft dabei die virtuellen Maschinen (VM) und *Container* im Griff zubehalten. *Nova* ist abhängig von folgenden anderen OpenStack-Modulen[12]:

- *Keystone* (Für Identity-Management)
- *Glance* (Für Images auf die *Nova* zugreifen kann)
- *Neutron* (Für Netzwerk)

Nova basiert auf der Script-Sprache *Python* mit einigen Anteilen von der *PHP-Template-Engine Smarty*[13]. Dadurch ist auch *Nova* in der Lage mittels *REST-API* auf Basis von *Python* angesprochen zu werden. Außerdem kann man verschiedene Tools (deutsch: Werkzeuge) an *Nova* andocken um auf *Nova*-Funktionen zuzugreifen. Dies kann über die Kommandozeile geschehen oder aber auch über das OpenStack-Dashboard *Horizon*. Die Kommunikation zwischen *Horizon* und *Nova* findet in diesem Fall ebenfalls über *REST-API* statt. Da *Nova* die Virtualisierungs-Bibliothek *libvirt* benutzt werden folgende *Hypervisoren* unterstützt:

- OpenVZ
- QEMU
- VirtualBox
- Vmware ESX
- Vmware Workstation/Player
- Xen
- Hyper-V
- PowerVM
- Virtuozzo
- Bhyve

Nova ist allerdings nicht auf die Virtualisierungsbibliothek *libvirt* beschränkt. Es kann mit einer Vielzahl von anderen Diensten kommunizieren. Beispielsweise die *Container*-Dienste *docker* oder *LXC* (*LXC* ist Bestandteil von *libvirt*). Bei *Container*-Diensten handelt es sich explizit nicht um einen *Hypervisor*. Desweiteren ist *Nova* in der Lage die Virtualisierungsschnittstelle vom *Linux-Kernel* zu benutzen (*KVM*).

Hypervisor

Der *Hypervisor* oder auch Virtual-Machine-Monitor (*VMM*) ist die Schlüsseltechnologie auf die *Nova*, als OpenStack Modul, aufbaut und dient zur Virtualisierung von ganzen Betriebssystemen. Dabei wird eine Abstraktionsschicht zwischen der Hardware, einem Host-Betriebssystem und mehreren Guest-Betriebssystemen gezogen. Das Host-Betriebssystem dient dabei als Wirt für die Guest-Betriebssysteme die nur als Gast auf dem Host-Betriebssystem laufen. Allgemeint wird dieser Vorgang „Virtualisierung“ genannt. Es ist allerdings anzumerken, dass diese Virtualisierung unterschiedlich stark verläuft. Je nach Hypervisor wird mehr oder weniger virtualisiert. So virtualisiert der *Hypervisor OpenVZ* beispielsweise nur auf Betriebssystemebene, bei *Xen* handelt es sich um eine *Paravirtualisierung* und der Hypervisor *KVM* virtualisiert komplett bis auf Kernel-Ebene (Der Kernel oder auch Betriebssystemkern kümmert sich um grundlegende Betriebssystembestandteile wie beispielsweise: Schnittstellen zu Hardware, Speicherverwaltung, Prozessverwaltung, Geräteverwaltung und Dateisysteme[13]). Bei der *Paravirtualisierung* wird das jeweilige Betriebssystem nur software-seitig virtualisiert[14]. Via *Xen*-virtualisierte Betriebssysteme haben also demnach keinen direkten Hardware-Zugang. *Xen* wird auch als *Hypervisor* für die *Amazon AWS* Cloud eingesetzt. In den meisten Fällen wird *KVM* für OpenStack *Nova* verwendet, dadurch können in den einzelnen virtuellen Maschinen verschiedene Betriebssysteme eingesetzt werden und eine Beschränkung auf nur ein Betriebssystem wie bei *OpenVZ* entfällt.

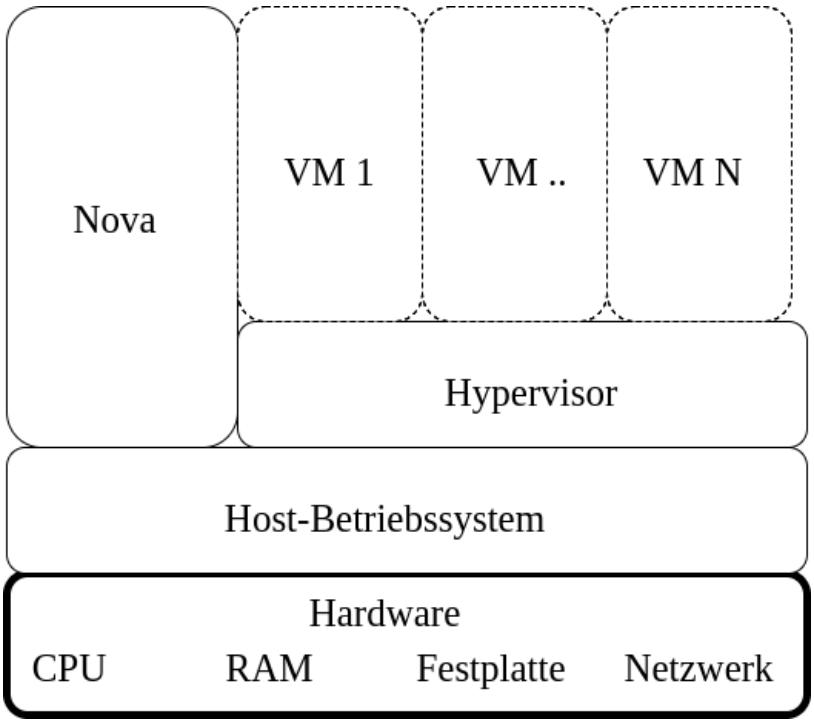

Abbildung 2: Zusammenspiel von Host-Betriebssystem und Hypervisor

Wie in der Abbildung 2 dargestellt ist ganz unten die Hardware-Ebene mit den verschiedenen *CPU*-Kernen, Festplatten und Netzwerk-Interfaces. Darüber liegt dann direkt das Host-Betriebssystem mit dem *KVM-Kernelmodul* und sonstigen *Kernel*-Bestandteilen die für die Virtualisierung wichtig sind (Dateisystem und Treiber). Als *Hypervisor* wird in der Abbildung 2 *QEMU* eingesetzt. *QEMU* virtualisiert die Hardware-Ressourcen. Über *QEMU* lassen sich dann Beschränkungen vornehmen (Zum Beispiel lassen sich darüber nur zwei *CPUs* von vier verfügbaren *CPUs* auf dem Host-Betriebssystem zuweisen). Über *iothreads* (Input/Output Threads) kommuniziert *QEMU* dann zwischen virtualisierten Hardware-Ressourcen und physischen Hardware-Ressourcen. In dem Guest-Betriebssystem kann dadurch der komplette *Kernel* virtualisiert werden. Dies hat den Vorteil, dass direkt auf *Kernelmodule* zugegriffen werden kann (beispielsweise *IPSec*-Module für *IPSec*-basierte *VPNs* oder *RNG*-Module um sichere Zufallszahlen direkt aus der Hardware zu ziehen).

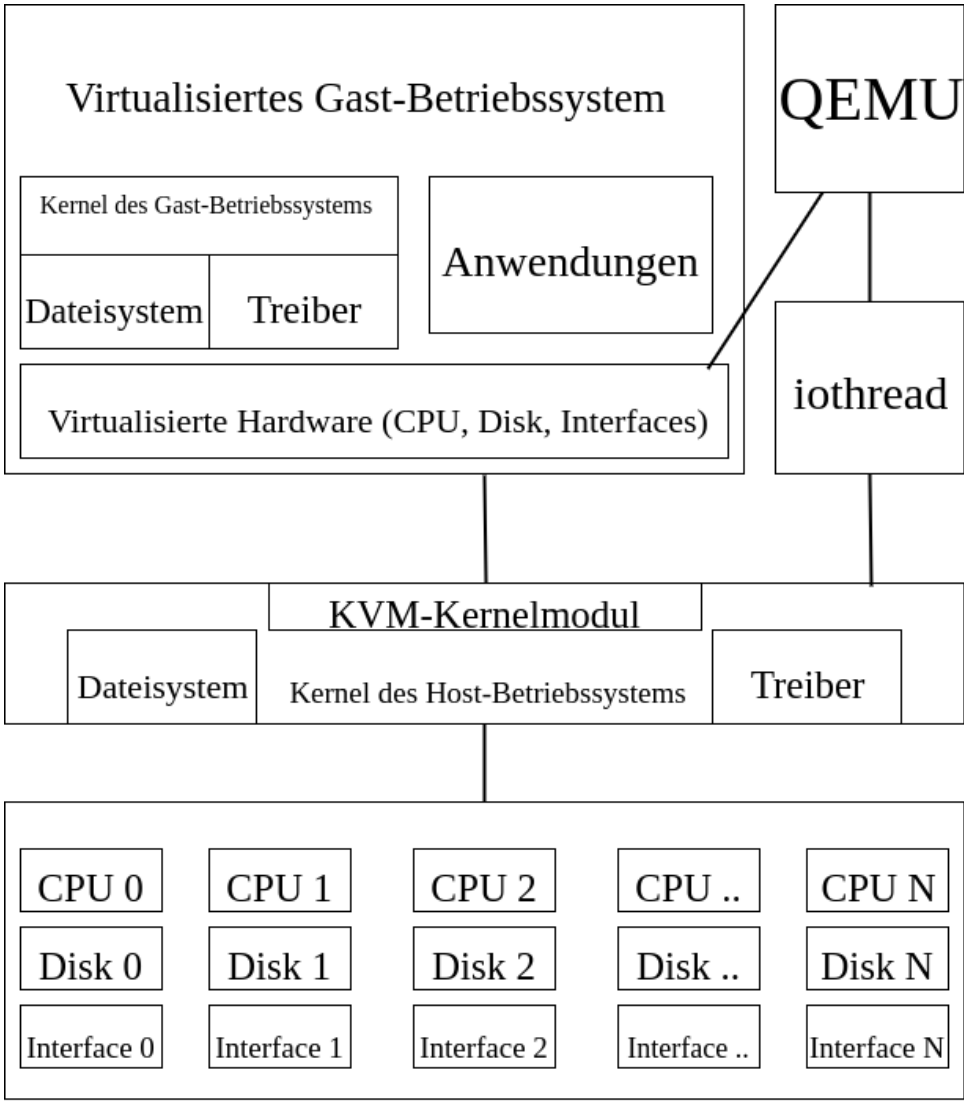

Abbildung 3: Vereinfachte Darstellung der KVM-Technologie

Nova-Systemarchitektur

Da es sich bei *Nova* um eine der größten Module für OpenStack handelt lassen sich *Nova*-Komponenten aus Performance-Gründen auch auf mehrere Hosts verteilen. Dadurch lässt sich eine *Nova*-Installation gezielt horizontal und vertikal skalieren. Vertikal skalieren in dem ein spezifischer Host mit einer Komponente mehr Hardware zugewiesen bekommt und horizontal skalieren in dem eine Komponente auf mehrere Hosts aufgeteilt wird. Dadurch wird eine gut kontrollierbare Lastverteilung erreicht. Dementsprechend nimmt die Kommunikation zwischen diesen einzelnen *Nova*-Komponenten, aber auch *Nova* externe Komponenten, eine besonders große Rolle ein. Für die Kommunikationen zwischen *Nova* eigenen Komponenten wird das Remote-Procedure-Call-Protokoll (*RPC*) verwendet. Als Basis für dieses Protokoll dient OpenStacks eigene *oslo.messaging*-Bibliothek. Um Informationen mit externen Komponenten wie beispielsweise *Keystone* auszutauschen wird eine *REST-API* verwendet. Desweiteren existiert eine *SQL*-Datenbank für diverse Daten die im laufenden Betrieb anfallen.

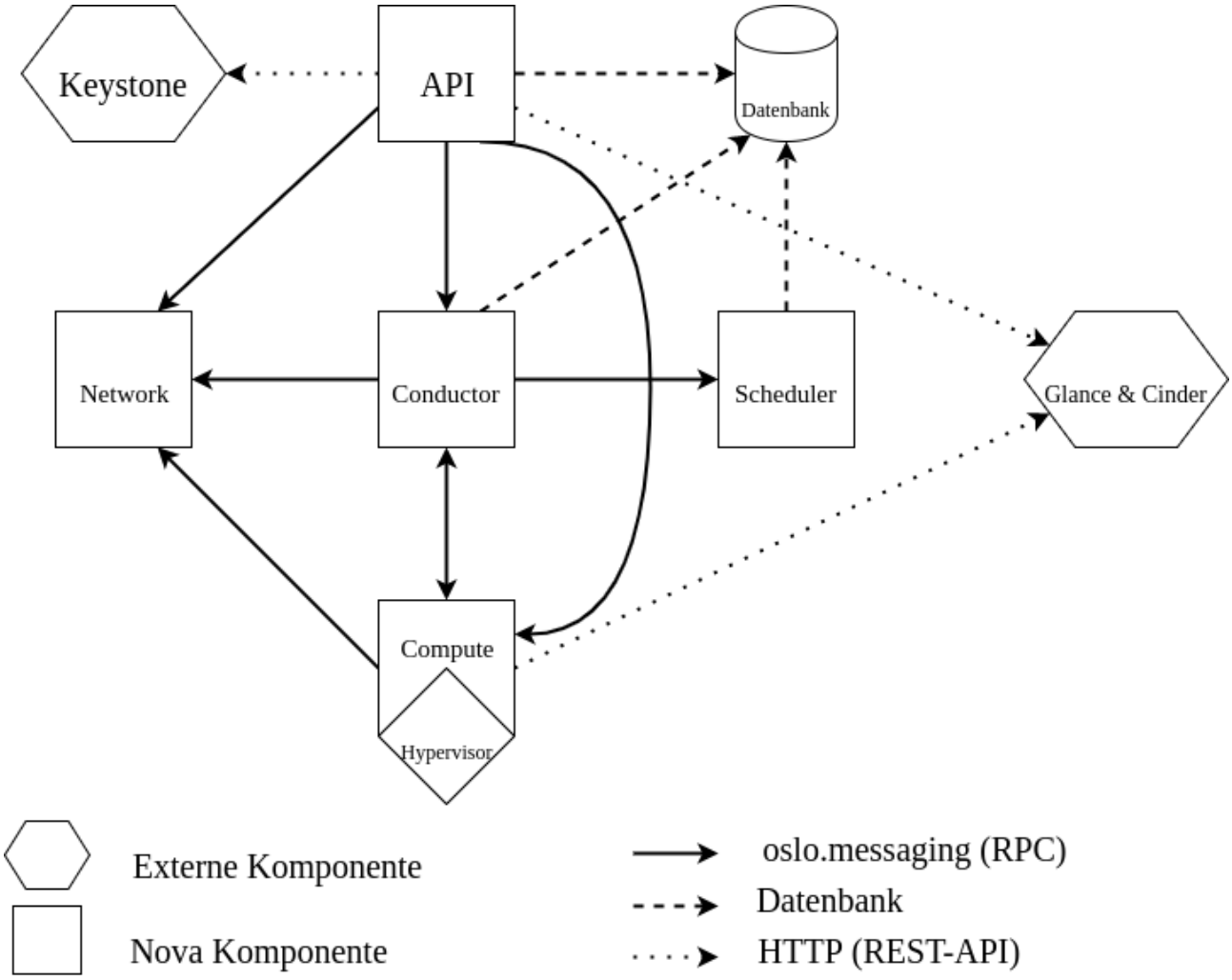

Abbildung 4: Nova-Systemaufbau ohne Neutron

Wie in Abbildung 4 verdeutlicht besteht *Nova* aus insgesamt vier Komponenten[15]:

- **Compute**
- **Conductor**
- **Scheduler**
- **Network**
- **API**

Jeder dieser Komponenten kommt eine bestimmte Aufgabe zu. Der **Scheduler** entscheidet auf welchem Host eine Instanz liegt. Dies ist besonders für die horizontale Skalierbarkeit wichtig. So sorgt der **Scheduler** dafür, dass alle Hosts immer gleich ausgelastet sind. Dementsprechend ist er vergleichbar mit einer intelligenten Lastverteilung für virtuelle Maschinen. Die **Compute**-Komponente ist das Bindeglied zwischen Hypervisor und den virtuellen Maschinen. De facto lassen sich über die **Compute**-Komponente neue virtuelle Maschinen erstellen, löschen und verschieben. Der **Conductor** wiederum managed Anfragen die einen Überblick über mehrere Komponenten benötigen. Dies ist zum Beispiel der Fall wenn man eine virtuelle Maschine nachträglich vergrößern oder verkleinern möchte. Dadurch das mehr Speicher frei oder belegt wird ändert sich wiederum die Balance zwischen den einzelnen Hosts. In diesem Fall würde der **Conductor** den **Scheduler** beispielsweise benachrichtigen, dass auf dem Host X nicht genug Speicher für eine Vergrößerung der virtuellen Maschine ist. Also würde der **Scheduler** dafür sorgen, dass die Maschine auf eine anderen Host mit genug freien Speicher wandert. Außerdem lässt sich über den **Conductor** die Datenbank ansprechen. Dies ist interessant für die **Compute**-Komponente. Die **Network**-Komponente wird eigentlich in gängigen Installationen von dem OpenStack-Modul *Neutron* ersetzt. Ist *Neutron* aber nicht vorhanden kommt diese **Network**-Komponente zum Einsatz, welche für die Vergabe von *IP*-Adressen, *VLANs* und *Bridges* zuständig ist. Die **API**-Komponente verbindet das OpenStack-Modul *Nova* mit anderen OpenStack-Modulen. Einkommende *HTTP*-Anfragen werden verarbeitet und für die anderen Teilkomponenten des *Nova*-Moduls in *RPC*-Anfragen umgewandelt. Für die *RPC*-Anfragen wird die OpenStack eigene *oslo.messaging*-Bibliothek verwendet. Die einkommenden *HTTP*-Anfragen werden via *REST-API* verarbeitet.

Desweiteren darf der Zusammenhang zwischen *Nova* und *Keystone*, *Glance*, *Cinder* oder *Swift*, und *Neutron* nicht unerwähnt bleiben. *Keystone* ist für die Benutzer-Berechtigungen zuständig, *Glance* für die Images, welche als *Template* für die virtuellen Maschinen dienen, *Neutron* für das Netzwerk und *Cinder* oder *Swift* für den Speicher. *Cinder* üblicherweise für als Image-Speicher und *Swift* als Speicher für die Daten von den virtuellen Maschinen.

Remote-Procedure-Call (RPC)

Für die Kommunikation zwischen Teilkomponenten eines OpenStack-Moduls wird die OpenStack *oslo.messaging*-Bibliothek benutzt[15]. Diese Bibliothek implementiert das *RPC*-Protokoll, welches eine Technik zur Realisierung von Interprozesskommunikation zur Verfügung stellt[16]. Interprozesskommunikation ist die Kommunikation zwischen zwei oder mehreren von sich unabhängigen Prozessen. Auf Betriebssysteme kann dies beispielsweise durch Signale oder Pipes geschehen. Pipes leiten die Ausgabe als Eingabe an ein anderes Programm weiter. Signale dagegen sind ein Interrupt auf der Prozessebene. *RPC* ist das Gegenstück dazu auf Netzwerk-Ebene. Durch *RPC* ist es möglich von einem Client aus Funktionen auf einem Server auszuführen. Auf das OpenStack Beispiel übertragen bedeutet dies, dass wenn in der *Nova*-Compute-Komponente eine virtuelle Maschine erstellt wird die Compute-Komponente mit einem *RPC*-Aufruf dies dem Scheduler mitteilen kann und eine Funktion im Scheduler aufruft damit die virtuelle Maschine auf einem Host mit freien Kapazitäten landet. Dazu implementiert die Anwendung einen **Stub** (deutsch: Stumpf). Dies ist ein Teil der Software der zwar auf dem einen Host erwähnt wird, aber auf diesem Host nicht implementiert ist. Die eigentliche Implementierung liegt auf einem anderen Host. Dieser **Stub** wiederum ist in der jeweiligen **Runtime Library** (deutsch: Laufzeit-Bibliothek) implementiert. Über diese Laufzeit-Bibliothek wird dann auch ein Netzwerk-Socket aufgebaut und ein Transport der Daten über das Netzwerk eingerichtet. Je nach Implementierung kann dies *UDP* oder *TCP* sein (in den meisten Fällen ist dies *UDP*).

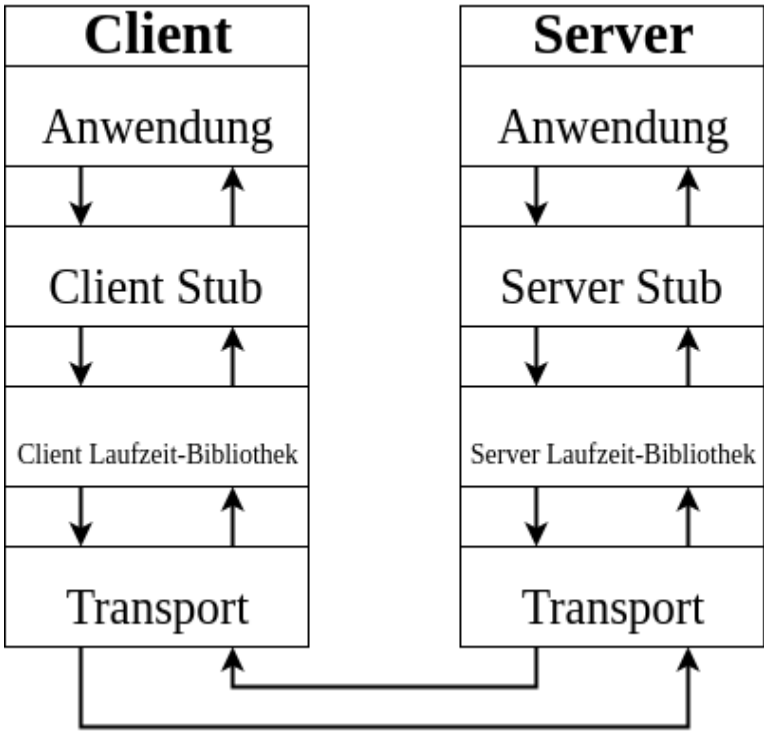

Abbildung 5: Vereinfachte Darstellung des Remote-Procedure-Call-Protokolls

Representational-State-Transfer (REST)

Für die Kommunikation zwischen den einzelnen OpenStack-Modulen kommt eine *REST-API* zum Einsatz. Das Programmierparadigma *REST* macht sich das Protokoll HTTP zu Nutze um eine *API* für Anwendungen über das Netzwerk bereitzustellen. Dabei werden die Anfragen und Antworten als *HTTP*-Requests definiert und Ressourcen werden anhand von einem Schlüssel-Wert-Verfahren auf dem Server abgelegt. Als Schlüssel dient eine eindeutige *URL* zu der jeweiligen Ressource. Die Ressource selbst wird häufig in Form von *JSON*, *XML* oder *HTML* abgelegt. Aber auch reiner Text ist möglich. Die wichtigsten vier *HTTP*-Requests sind[17]:

- **GET**: Holt Informationen vom Server ohne den Zustand am Server zu ändern.
- **POST**: Legt eine neue Sub-Ressource unter einer bereits existierenden Ressource an.
- **PUT**: Legt eine Ressource an. Falls diese existiert wird sie geändert.
- **DELETE**: Löscht die angegebene Ressource vom Server.

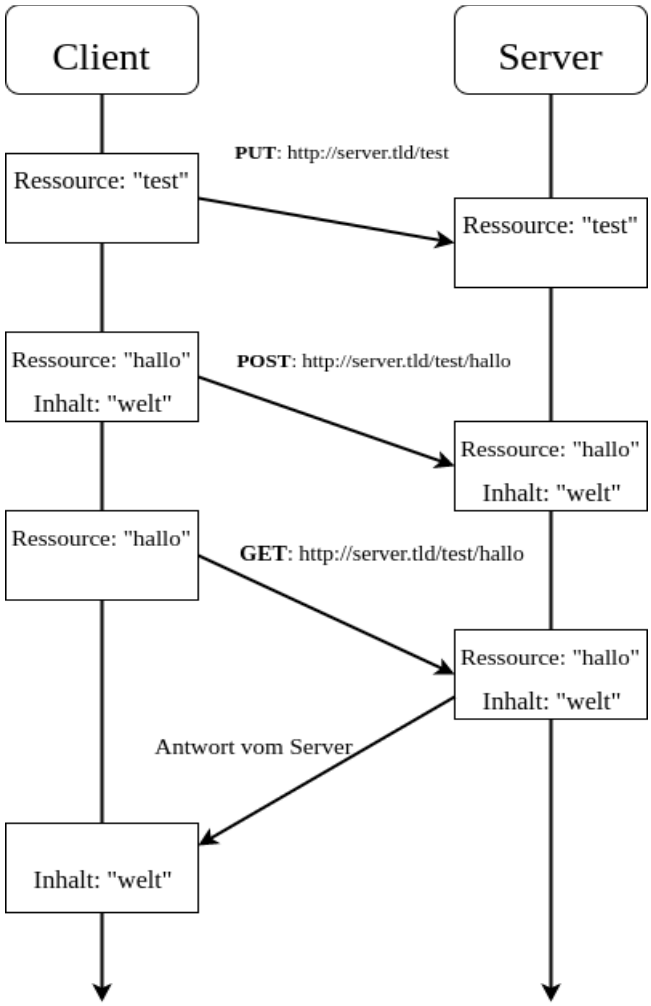

Abbildung 6: Stark vereinfachtes Beispiel von REST

Die Abbildung 6 zeigt ein stark vereinfachtes Beispiel einer *REST-API.* Es wird demonstriert wie zu erst mit dem **PUT**-Request eine neue Ressource angelegt wird. Danach wird mit dem **POST**-Request die Ressource „hallo“ mit dem Inhalt „welt“ auf dem Server angelegt. Dieser angelegte Wert wird dann mit dem *HTTP*-Request **GET** abgefragt und der Client bekommt als Antwort „welt“ zurück. Ein **GET**-Request kann mit der Sprache *Python* zum Beispiel wie folgt aussehen:

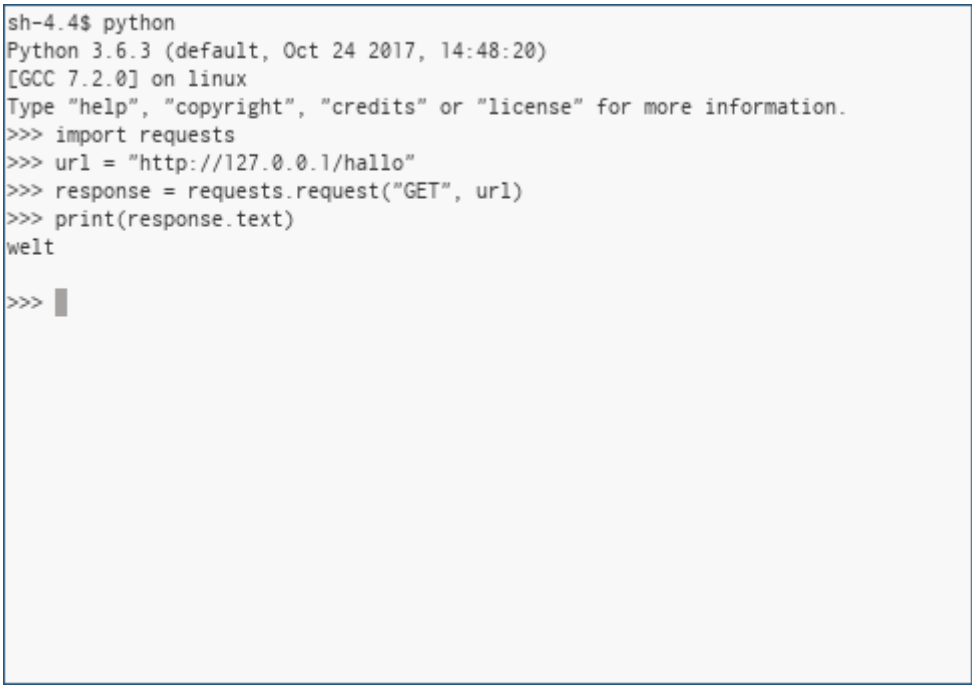

Abbildung 7: Beispiel eines GET-Requests via Python

Neutron

Die Frage um die Skalierbarkeit und Emulation von Netzen ist bereits seit 1997 ein wichtiges Thema in der Informatik[18], welches dementsprechend auch für das Cloud-Computing wichtig ist. Um so ein skalierbares Netzwerk mit OpenStack aufzuziehen wird das Modul *Neutron* benutzt. *Neutron* besteht aus folgenden Komponenten:

- **neutron-server**: Diese Komponente läuft auf einem dedizierten Netzwerkknoten und bietet unteranderem die *API* an.
- **neutron-*-plugin**: Diese Komponente sind plugins (deutsch: Erweiterungen) für den neutron-server.
- **neutron-*-agent**: Diese Komponente läuft auf allen *Nova*-Compute-Knoten um die lokalen virtuellen Switch Konfigurationen anzupassen. Auch Plugin Agent genannt.
- **neutron-dhcp-agent**: Bietet *DHCP* für die einzelnen virtuellen Maschinen auf den *Nova*-Compute-Knoten an. Das Dynamic-Host-Configuration-Protocol (*DHCP*) dient der Vergabe von IP-Adressen an die virtuellen Maschinen[19].
- **neutron-l3-agent**: Bietet *NAT* auf Layer 3 des *OSI*-Schichtenmodells für externen Netzwerkzugang für die virtuellen Maschinen an. *Network Address Translation* wird dazu benutzt um mehrere Netze miteinander zu verbinden. Dabei kommen mehrere private *IP*-Adressen zum Einsatz[20].
- **SDN server/services**: Bietet zusätzliche Netzwerkdienste an. *SDN* steht für Software-Defined Networking und ist ein seit 2005 gebräuchlicher Ansatz zum Bau von Computernetz-Geräten und Software[21]. Dabei wird das Netzwerk in zwei Ebenen abstrahiert. Die Control-Plane und die Data-Plane. Die Control-Plane bestimmt den Bestimmungsorten für Daten und die Data-Plane leitet diese ausgewählten Daten dann weiter. Dadurch ist es möglich ganze Netzwerke auf Softwareebene zu abstrahieren und diese „programmierbar“ zu machen. Ein manuelles Eingreifen in den Ablauf ist damit nicht mehr nötig und die Effizienz wird gesteigert.

In der nachfolgenden Abbildung wird der interne Aufbau von *Neutron* grafisch vermittelt. Daraus wird sofort ersichtlich, dass auch *Neutron* auf die zwei Konzepte *REST* und *RPC* zur Kommunikation setzt. So werden die *Neutron* eigenen Agents via *RPC* angesprochen und der *SDN-Service* via *REST*:

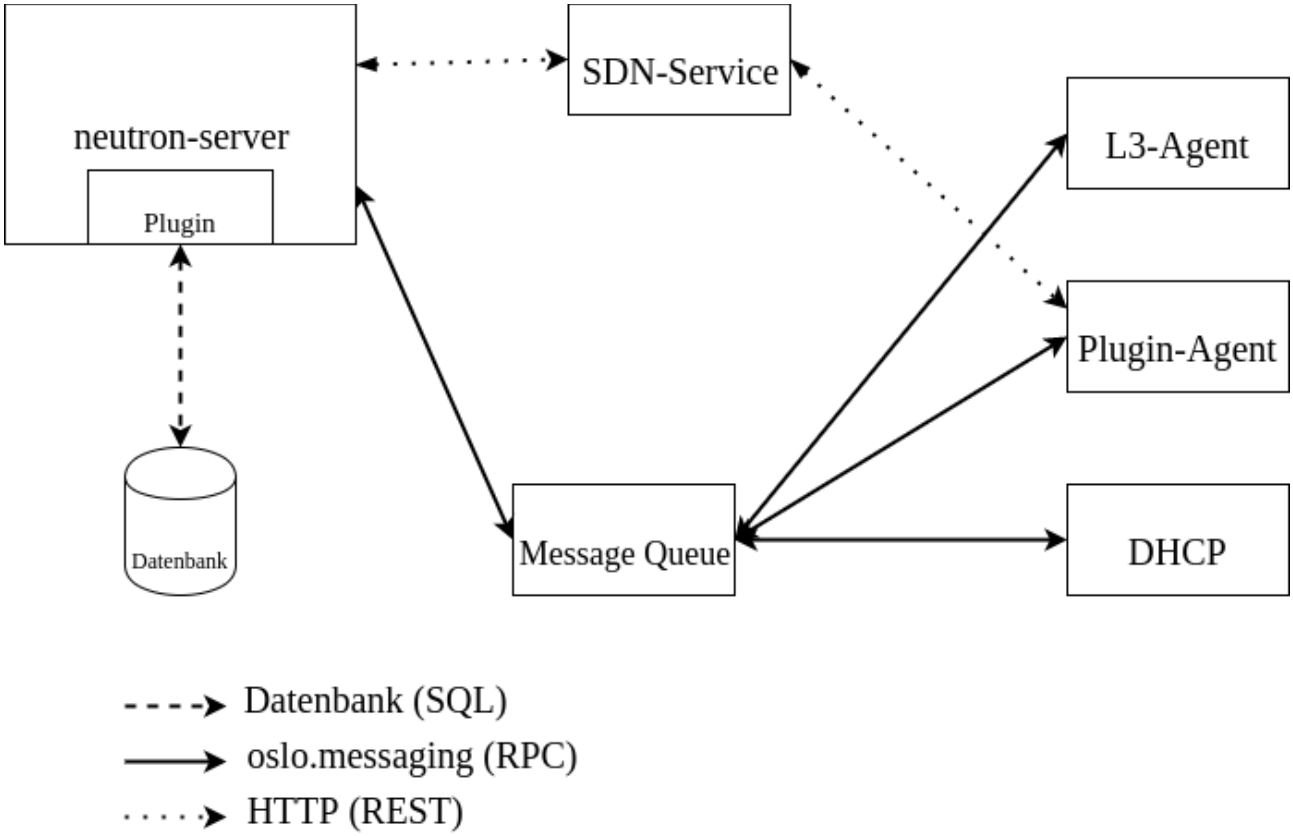

Abbildung 8: Interner Aufbau des Neutron-Moduls

Technisches Fazit

OpenStack baut auf bekannte Technologien und weiß diese gezielt miteinander zu verknüpfen. Für die Kommunikation werden die seit Jahren bewährten Technologien *RPC* und *REST* eingesetzt, welche sich auszeichnen durch einen einfachen Aufbau und der Möglichkeit zur weiteren Skalierbarkeit. Außerdem werden als Datenspeicher *SQL*-Datenbanken verwendet, welche sich ebenfalls seit Jahren bewährt haben und einen sicheren Platz zum Abspeichern von Daten darstellt, welcher gleichzeitig mit der Möglichkeit von schnellen Zugriffen und horizontaler Skalierbarkeit überzeugt. Dennoch schafft OpenStack es auch neue Technologien zu verwenden. So baut der Großteil des Projekts auf die noch im Vergleich zu *C* oder *C++* recht junge Sprache *Python* und auf deren *Frameworks* und Bibliotheken. Für das Netzwerk ist *Software-Defined Networking*von zentraler Bedeutung. Nur durch *Software-Defined Networking* ist es möglich eine Cloud mit mehreren 10.000 virtuellen Maschinen richtig zu managen und das Netzwerk dabei in einem stabilen und dennoch leicht veränderbaren Zustand zubehalten. Durch *Software-Defined Networking* sind die Besitzer einer solchen Cloud in der Lage ganze Netzwerke *on-demand* auszurollen und sie bei Nicht-Bedarf auch wieder verschwinden zu lassen ohne, dass auch nur eine Person einen Finger dafür rühren muss. Der Trend geht also weiter in Richtung Automatisierung, Abstraktion und Algorithmen, welche intelligenter werden je mehr man sie einsetzt. Der nächste Schritt wäre demnach eine Verknüpfung von Cloud-Technologien mit selbstlernenden Algorithmen um den Faktor Mensch komplett abzuschaffen und zumindest das Skalieren in die Hände von Algorithmen zulegen.

Nachwort

Der interne Aufbau einer OpenStack Cloud ist so komplex, dass dieses Dokument eigentlich ein ganzes Buch ausfüllen müsste. Ich hätte mich gerne noch weiter mit den einzelnen Modulen beschäftigt, insbesondere den Feinheiten von *Neutron*. Die Liste lässt sich natürlich beliebig erweitern um andere interessante Module wie *Glance* oder *Swift*. Ebenfalls interessant wäre ein Einblick in moderne Deployment-Strategien gewesen. Deployment umschreibt den Vorgang der Installation von OpenStack. So betreut das OpenStack-Entwicklerteam einige Ansible-Module für OpenStack um OpenStack mit einem Druck auf die Entertaste in ganze Rechenzentren zu deployen[22]. Ansible ist ein Deployment-Werkzeug um Installation in eine einfache Syntax zu abstrahieren. Dies unterstreicht nochmal die Komplexität von OpenStack, die Installation scheint so komplex zu sein, dass nun damit begonnen wird auch die Installation zu abstrahieren. Daraus folgt die Frage ob so ein komplexer Aufbau überhaupt nötig ist oder ob ein weniger komplexer Aufbau nicht ausreichend wäre. Die Frage lässt sich durch den Verweis auf unterschiedliche Anforderungen beantworten. Für kleinere Rechenzentren-Betreiber wie beispielsweise der TU-Clausthal wäre eine OpenStack-Installation schlichtweg zu komplex, da alleine für die Wartung eine mehrere Mitarbeiter eingespannt werden müssten die nichts anderes tun. Für größere Betreiber wie Amazon oder Google jedoch, mit Kunden die auf Skalierbarkeit und dynamisches Verhalten angewiesen sind, ist so eine Cloud jedoch perfekt. Was durchaus denkbar wäre, wäre eine deutschlandweite akademische Cloud in der sich die Universitäten Rechenpower und Festplattenspeicher gezielt je nach Bedarf einkaufen können.

Glossar

NIST	National Institute for Standards and Technology
API	Application Programming Interface. Eine Schnittstelle für Programmiersprachen um auf ein Programm extern zuzugreifen.
Fork	Ein Fork ist eine Abspaltung oder ein Entwicklungszweig eines Projektes in mehrere Folgeprojekte
Web-Hoster	Ein Web-Hoster bietet Web-Infrastruktur zum Kauf oder zur Miete an.
NASA	National Aeronautics and Space Administration
Deployment Models	Verschiedene Einsatzbereiche. In dieser Seminararbeit sind Einsatzbereiche einer Cloud gemeint.
Service Models	Service Models umschreiben das Verhältnis zwischen Kunden, den Möglichkeiten der Cloud und dem Cloud-Anbieter.
IaaS	Infrastructure as a Service. Ein *Service Model* bei dem Kunden sich Rechenzeit einfach on-demand virtuell anmieten können, anstatt in teure eigene Hardware zu investieren.
OpenMPI	Ein Framework zum verteilten Rechnen in Cloud-Umgebungen.
CPU	Central-Processing-Unit. Gemeinhin auch bekannt als Prozessor.
RAM	Random-Access Memory. Ein flüchtiger aber sehr schnell zugreifbarer Speicher.
IEEE	Institute of Electrical and Electronics Engineers
UNIX	Ein Von Bell Laboratories entwickeltes Mehrbenutzer-Betriebssystem für Computer
Horizon	Das OpenStack Dashboard
Python	Eine Interpreter-basierte Script-Sprache
Javascript	Eine Interpreter-basierte Script-Sprache. Im Syntax Java ähnlich. Weitverbreitet für dynamischen Web-Content.
HTML	Hyper-Text-Markup-Language. Auszeichnungssprache zur Darstellung von Webseiten.
CSS	Cascada-Style-Sheet. Auszeichnungssprache zur Gestaltung von Webseiten. Als Ergänzung zu HTML gedacht.
URL	Uniform Resource Locator. Eine eindeutige für Menschen merkbare Adresse für Webseiten.
REST-API	Representational State Transfer (REST)-API. Basiert

	auf dem Senden von HTTP-Requests.
HTTP	Hypertext-Transfer-Protocol. Das gängige Protokoll zum Aufrufen von Webseiten auf Port 80.
HTTPS	HTTP nur mit Verschlüsselung via TLS (ehemals SSL).
TLS	Transport Layer Security. Hybrides Verschlüsselungsprotokoll.
SSL	Secure Sockets Layer. Siehe auch TLS.
Corporate Design	Das Design einer Firma. Logos, Slogans, etc.
Nova	Das OpenStack Modul für das Management der Container und virtuellen Maschinen.
Hypervisor	Abstraktionsschicht zwischen der Hardware um ein oder mehrere Betriebssysteme auf der selben Hardware zu benutzen.
Keystone	OpenStack Modul zur Verwaltung von Benutzer-Identitäten/
Glance	OpenStack-Modul zur Verwaltung von fertigen Images die als Basis für *Nova* dienen.
Neutron	OpenStack-Modul zur Verwaltung von Netzwerk-Ressourcen (beispielsweise die Vergabe von IP-Addressen)
PHP	PHP Hypertext Processor. Serverbasierte Script-Sprache.
Template-Engine	Eine Software um Text-Templates anzupassen. Beispielsweise um nachträglich Werte in Dateien einzufügen.
Smarty	Eine auf *PHP* basierte *Template-Engine*.
Libvirt	Quelloffene Virtualisierungsbibliothek für Linux-basierte Systeme.
Container	Ein vom restlichen System isoliertes Verzeichnis was quasi pseudo-virtualisiert wird. In Linux geschieht diese Isolation mit Cgroups und Namespaces.
Docker	Populärer Container-Dienst
VMM	Virtual-Machine-Monitor
Kernel	Betriebssystemkern, welcher sich um grundlegende Betriebssystemaufgaben kümmert beispielsweise: Dateisysteme.
Amazon AWS	Amazon Web Services. Eine große kommerzielle Public-Cloud von Amazon.
Linux	Ein von Linus Torvalds geschriebens Betriebssystem. Sehr nah verwandt mit UNIX.
Kernelmodul	Optionale Software-Module die den Betriebssystemkern erweitern um zusätzliche Funktionen. Performanter und sicherer als auf Applikationsebene.

Iothreads	Eingabe/Ausgabe-Schnittstellen zwischen zwei Endpunkten. Hier zwischen QEMU und Host-Betriebssystem
IP	Internet-Protocol.
IPSec	Internet Protocol Security. Erweiterung für das Internet Protocol um unsichere IP-Netze abzusichern.
RNG	Random-Number-Generator. Zufallszahlengenerator um sichere Zufallszahlen für kryptische Verfahren bereitzustellen.
VPN	Virtual-Private-Network. Virtuelles in sich geschlossenes Netz.
RPC	Remote-Procedure-Call. Protokoll zum Aufrufen von entfernten Funktionen.
Oslo.messaging	OpenStacks eigene RPC-Implementierung.
SQL	Structured-Query-Language. Eine Beschreibungssprache für Datenbanken.
VLAN	Virtual-Local-Area-Network. Logisches Teilnetz innerhalb von physischen Netzen.
Bridge	Eine Bridge verbindet zwei Segmente auf dem OSI-Layer 2.
OSI-Modell	Open System Interconnection Model. Sieben Schichten Modell für Computernetze.
UDP	User-Datagram-Protocol. Zustandsloses Protokoll für das Netzwerkverbindungen
TCP	Transmission-Control-Protocol. Zustandsbehaftetes Protoll für Netzwerkverbindungen
JSON	JavaScript Object Notation. Kompaktes Objektorientiertes Datenformat zum Speichern von Informationen
XML	Extensible Markup Language. Beschreibungssprache zum strukturieren von Daten.
DHCP	Dynamic Host Configuration Protocol. Dient der dynamischen Vergabe von IP-Adressen an Hosts.
NAT	Network Address Translation. Verbindet zwei Netzwerksegmente miteinander durch das Erstellen von eigenen privaten Adressräumen.
SDN	Software-Defined Networking. Ein seit 2005 gebräuchlicher Ansatz zur Verwaltung von Netzen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Peter Mell und Timothy Grance. „The NIST Definition of Cloud Computing“. Special Publication 800-145. The National Institute of Standards and Technology. September 2011. URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (besucht am 19.11.2017)

[2] „OpenStack Release Page“. URL: https://wiki.openstack.org/wiki/Releases (besucht am 19.11.2017)

[3] „Rackspace OpenStack“. URL: https://www.rackspace.com/de-de/openstack (besucht am 19.11.2017)

[4] „Rackspace acquires Anso Labs“. URL: https://blog.rackspace.com/rackspace-acquires-anso-labs-furthers-commitment-to-openstack (besucht am 19.11.2017)

[5] „Open Telekom Cloud“. URL: https://www.openstack.org/marketplace/public-clouds/deutsche-telekom/open-telekom-cloud (besucht am 19.11.2017)

[6] „The OpenStack Mission“. URL: https://www.openstack.org/marketplace/public-clouds/deutsche-telekom/open-telekom-cloud (besucht am 19.11.2017)

[7] Jim Gray und Daniel P. Siewiorek. „High-Availability Computer Systems“, Digital Equipment Corporation und Carnegie Mellon University, für die IEEE, URL: http://research.cs.wisc.edu/areas/os/Qual/papers/ha-systems.pdf (besucht am: 24.11.2017)

[8] Douglas McIlroy, E.N. Pinson und B.A. Tague. „The Bell System Technical Journal“. „Unix-Time-Sharing System: Forward“. Bell Laboratories. 8. Juli 1978. Seite 1902-1903. URL: http://emulator.pdp-11.org.ru/misc/1978.07_-_Bell_System_Technical_Journal.pdf (besucht am 23.11.2017)

[9] „OpenStack Horizon Quellcode“. URL: https://github.com/openstack/horizon (besucht am 24.11.2017)

[10] „Writing your first Django app, part1“. URL: https://docs.djangoproject.com/en/1.11/intro/tutorial01/ (besucht am 24.11.2017)

[11] „The Horizon Module“. URL: https://docs.openstack.org/horizon/latest/contributor/ref/horizon.html (besucht am 24.11.2017)

[12] „The Nova Module“. URL: https://docs.openstack.org/nova/latest/ (besucht am 27.11.2017)

[13] „Kernel (Betriebssystem). URL: https://de.wikipedia.org/wiki/Kernel_(Betriebssystem) (besucht am 03.12.2017)

[14] Christian Rebischke. „Amazon Web Services: EC2-virtual-server und EC2-container“. Technische Universität Clausthal. Seite 9. 8. Oktober 2017. URL: https://github.com/shibumi/aws-ec2-project-paper/releases/download/final-release/essay.pdf (besucht am 03.12.2017)

[15] „Nova System Architecture“. URL: https://docs.openstack.org/nova/pike/user/architecture.html (besucht am 04.12.2017)

[16] „Remote Procedure Call“. URL: https://de.wikipedia.org/wiki/Remote_Procedure_Call (besucht am 04.12.2017)

[17] „Representational State Transfer“. URL: https://de.wikipedia.org/wiki/Representational_State_Transfer (besucht am 04.12.2017)

[18] Harald Richter. „Verbindungsnetzwerke für parallele und verteilte Systeme“. Spektrum Akademischer Verlag. Heidelberg. Seite 95. 1997. ISBN: 3827401879. URL: https://www.in.tu-clausthal.de/fileadmin/internes/Richter_Buch_1997.pdf (besucht am 04.12.2017)

[19] „Dynamic Host Configuration Protocol“. URL: https://de.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol (besucht am 04.12.2017)

[20] „Network Address Translation“. URL: https://de.wikipedia.org/wiki/Netzwerkadress%C3%BCbersetzung (besucht am 04.12.2017)

[21] „Software-defined networking“. URL: https://de.wikipedia.org/wiki/Software-defined_networking (besucht am 04.12.2017)

[22] „Openstack-Ansible“. URL: https://github.com/openstack/openstack-ansible (besucht am 04.12.2017)