

Kernel as a Service - Maßschneidung von Kernel für Infrastruktur-Clouds

Nico Weichbrodt

7. November 2013

Technische Universität Braunschweig
Institut für Betriebssysteme und
Rechnerverbund

Bachelorarbeit

Kernel as a Service - Maßschneidung von
Kernel für Infrastruktur-Clouds

von
Nico Weichbrodt

Aufgabenstellung und Betreuung:

Prof. Dr. R. Kapitza

Braunschweig, den 7. November 2013

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, den 7. November 2013

Unterschrift

Kurzfassung

In dieser Bachelorarbeit wird *Kernel as a Service* vorgestellt, ein System zum Erstellen von maßgeschneiderten Kernen in einer Cloud Umgebung. Basierend auf der Cloud Infrastruktur *OpenStack* und dem Kernelanalysepaket *undertaker* dient Kernel as a Service dazu die Sicherheit von virtuellen Maschinen zu erhöhen. Dies wird erreicht, indem an individuelle Bedürfnisse angepasste, maßgeschneiderte Linux Kernel erstellt werden. Kernel as a Service integriert sich dazu in OpenStack Horizon und bietet Benutzern eine einfaches Interface zum Erstellen von maßgeschneiderten Kernen. Ein maßgeschneiderter Kernel erreicht eine Verringerung der Angriffsfläche von 50% bis 85% gegenüber einem generischen Kernel ohne dabei die Leistung der angebotenen Dienste zu beeinflussen.

Abstract

This bachelor thesis introduces *Kernel as a Service*, a system for creating tailored Linux kernels in a cloud environment. Based on the infrastructure cloud *OpenStack* and the kernel analysis programm package *undertaker*, Kernel as a Service is used to increase the security of virtual machines. This is achieved by creating tailored Linux kernels, which are customized to the users needs. Kernel as a Service integrates into OpenStack Horizon and offers users a simple interface for creating tailored kernels. A tailored kernel has a reduced attack surface of 50% to 85% compared to a generic kernel without impacting its performance.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Struktur dieser Bachelorarbeit	2
2	Basistechnologien	3
2.1	Infrastructure as a Service	3
2.2	OpenStack	4
2.2.1	Komponenten und Aufbau	4
2.3	Undertaker	7
2.3.1	Bereitgestellte Werkzeuge und deren Benutzung	7
3	Kernel as a Service	13
3.1	KaaS Horizon Modul	13
3.2	KaaS Buildserver	15
3.3	KaaS Nutzer VM	15
4	Aufbau und Implementierung	17
4.1	Gemeinsam verwendeter Basis Server	17
4.2	KaaS Buildserver	18
4.2.1	Ablauf des Kernelbaus	20
4.3	KaaS Nutzer VM	21
4.4	Eigenbau der VM Images	21
4.5	KaaS Horizon Modul	22
4.6	Zusammenspiel der Komponenten	27
4.6.1	KaaS intern	28
5	Probleme während der Entwicklung	31
5.1	Undertaker konnte keine valide Kernelkonfiguration erstellen	31
5.2	Kernelinstallation in einer <code>chroot</code> Umgebung	31
5.3	Aufhängen beim Boot durch OpenStack	32
6	Evaluation	35
6.1	Ergebnisse der Performanzanalyse	36
6.2	Vergleich eines maßgeschneiderten und generischen Kernels	37
7	Ähnliche Ansätze und mögliche Erweiterungen	41
7.1	Kernel verkleinern mit Linux Boardmitteln	42

Inhaltsverzeichnis

7.2	Mögliche Erweiterungen	43
7.2.1	CVE Alarme	44
8	Zusammenfassung	45
	Literaturverzeichnis	46

Abbildungsverzeichnis

2.1	Generischer Aufbau von IaaS Clouds	3
2.2	Aufbau und Zusammenspiel der OpenStack Komponenten	5
2.3	Zustandsautomat für das Tracen von Funktionen im Kernel	9
3.1	Anwendungsfalldiagramm für die beiden Nutzertypen von KaaS. Für OpenStack wurden nicht alle möglichen Anwendungsfälle an- gegeben.	14
4.1	Sequenzdiagramm für eine Client-Server Verbindung mit der <code>Connection</code> Klasse	18
4.2	Zustandsautomat für einen Kernelbau	20
4.3	Übersichtsbereich des KaaS Horizon Moduls. Sichtbar für alle Be- nutzer	24
4.4	Zwischenseite auf dem Weg zum Kernelbau. KaaS empfiehlt mit dem Kernelbau zu warten	24
4.5	Zwischenseite auf dem Weg zum Kernelbau. KaaS empfiehlt, dass mit dem Kernelbau begonnen werden kann	25
4.6	Administrativer Verwaltungsbereich des KaaS Horizon Moduls. Sichtbar nur für Administratoren	26
4.7	Formular zum Einstellen welche Images KaaS Nutzer VMs sind	27
4.8	Formular zum Starten von KaaS Buildservern	27
4.9	Interner Ablauf eines Kernelsbaus in KaaS	29
6.1	Testergebnisse aller VMs im direkten Vergleich; weniger ist besser	37
6.2	Verringerung der kompilierten Quellcodedateien für den maßge- schneiderten Kernel im Vergleich zum Ubuntu Standard Kernel für einen LAMP Server Quelle: [1]	38
6.3	Verringerung der kompilierten Quellcodedateien für den maßge- schneiderten Kernel im Vergleich zum Ubuntu Standard Kernel für einen memcached Server	39

Tabellenverzeichnis

4.1	Statuscode eines Kernelbaus und deren Bedeutung	19
6.1	Tastdauer des <code>memcslap</code> Benchmarks für ein unverändertes Ubuntu 12.04. Angabe in Sekunden; weniger ist besser	36
6.2	Tastdauer des <code>memcslap</code> Bechmarks für eine KaaS Nutzer VM (basierend auf Ubuntu 12.04). Angabe in Sekunden; weniger ist besser	36
6.3	Tastdauer des <code>memcslap</code> Bechmarks für eine VM mit maßgeschneidertem Kernel in einem Ubuntu 12.04. Angabe in Sekunden; weniger ist besser	37
6.4	Anzahl der kompilierten Quellcodedateien für verschiedene Bereiche des Linuxkernels	39

1 Einleitung

Infrastructure as a Service (IaaS) Anbieter wie Amazon oder Rackspace möchten es ihren Kunden einfach machen einen schnellen Einstieg zu finden. Dazu werden Templates von virtuellen Maschinen bereitgestellt. Dies sind fertige Images, die meistens auf einer bekannten Linux Distribution wie Ubuntu¹ oder CentOS² basieren. Diese Images enthalten ein Minimalsystem und den Linux Kernel. Da die Anbieter der Distributionen nicht wissen können für welchen Zweck ihre Distribution genutzt wird, ist der verwendete Kernel auf größtmögliche Kompatibilität ausgelegt. Dies birgt zunächst gewisse Vorteile, denn je mehr Features der Kernel hat, desto mehr kann der Nutzer seine individuellen Vorhaben umsetzen. Dies ist zwar richtig, jedoch werden die virtuellen Maschinen mit ihren featurereichen Kernels mittlerweile auf eine andere Art und Weise genutzt.

Es hat sich der Trend zu spezialisierten VM gezeigt. Früher wurden von einem Server mehrere Dienste bereitgestellt. Dies wird nun von mehreren virtuellen Maschinen übernommen, indem jede virtuelle Maschine einen Dienst anbietet. Dies bedeutet auch, dass viele Teile des Kernels gar nicht benutzt werden. Gerade selten benutzte Kernelfeatures können Sicherheitslücken enthalten, die es Angreifern ermöglichen in Systeme einzudringen. Verhindern lässt sich dies am Besten, indem unbenötigte Kernelfeatures gar nicht erst im Kernel enthalten sind. Um dies zu erreichen, müssten Nutzer sich eigene Kernel bauen, die auf ihre individuellen Bedürfnisse zugeschnitten sind.

Leider ist der Prozess zum Erstellen eines maßgeschneiderten Kernels nicht trivial. Die Konfiguration des Linux Kernels besteht aus über 11.000 verschiedenen Optionen, die in komplexen Abhängigkeitsbeziehungen stehen. Für Benutzer ist es daher nicht einfach genau die Optionen zu finden, die benötigt werden. Tools sollen dabei helfen die richtigen Optionen zu finden, aber trotzdem ist der Prozess des Kernelbauens weiterhin ein Vorgang, der von Hand erledigt werden muss. Aus Zeitgründen und Unwissen wird ein maßgeschneiderter Kernel daher so gut wie nie verwendet. Eine vereinfachte Bedienung in Form von Automatisierung des Kernelbauprozesses wäre daher sinnvoll, um Kernelmaßschneidern für Anwender interessant zu machen.

Im Rahmen dieser Bachelorarbeit wurde daher *Kernel as a Service* (KaaS) entwickelt. Ein System um maßgeschneiderte Kernel für den Nutzer vollautomatisiert zu erstellen. KaaS baut dafür auf *OpenStack*[2], einer quelloffenen IaaS Cloud

¹<http://www.ubuntu.com/>

²<http://www.centos.org/>

1 Einleitung

Plattform, und *undertaker*[\[3\]](#), einem Programmpaket zum Kernel Maßschneidern, auf. Durch die Nutzung von *undertaker* ist es KaaS möglich die Angriffsfläche des verwendeten Linux Kernels um 50% bis 85% zu verringern.

1.1 Struktur dieser Bachelorarbeit

Im **zweiten Kapitel** werden die Basistechnologien OpenStack und *undertaker* angesprochen. Es wird der Aufbau von OpenStack und die Arbeitsweise von *undertaker* erklärt. **Kapitel drei** widmet sich dem ursprünglichen Design von *Kernel as a Service*. Die Implementierung folgt in **vierten Kapitel**. Gemeinsamkeiten und Unterschiede zum Design werden dargelegt und die Software wird vorgestellt. Über Probleme während der Entwicklung wird in **Kapitel fünf** berichtet. Eine Evaluation der erstellten Kernel findet im **sechsten Kapitel** statt. Am Beispiel von *memcached* wird eine Performanzevaluation zwischen einem generischen und einem maßgeschneiderten Kernel erstellt. Über verwandte Arbeiten und ähnliche Methoden wird im **siebten Kapitel** berichtet. Letztendlich wird im **achten Kapitel** eine Zusammenfassung gegeben und ein Zukunftsausblick bezüglich möglichen Erweiterungen von Kernel as a Service gemacht.

2 Basistechnologien

Kernel as a Service ist kein eigenständiges Projekt. Vielmehr baut es auf vorhandenen Technologien und Programmen auf und verbindet diese mit eigener Software zu einem Paket. Diese Basistechnologien sollen in den folgenden Abschnitten vorgestellt werden. Die Verbindung zu KaaS folgt dann in Kapitel 3.

2.1 Infrastructure as a Service

Cloud Computing dient als Überbegriff für drei Gebiete: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) und *Software as a Service* (SaaS). Wird von Cloud Computing gesprochen, ist in den meisten Fällen Infrastructure as a Service gemeint. IaaS beschreibt das Konzept, dass IaaS Anbieter Ihren Kunden Infrastruktur in Form von Computern und deren Vernetzung zur Verfügung stellen, welche im Normalfall virtualisiert sind. Dem Nutzer wird dabei vom IaaS Anbieter scheinbar unbegrenzte Ressourcen zur Verfügung gestellt, die er jederzeit nutzen kann. Typisch für IaaS Anbieter ist die Abrechnungsmethode welche als Pay-as-you-go bezeichnet wird. Abgerechnet wird dabei die Zeit, die die virtuellen Maschinen eingeschaltet sind.

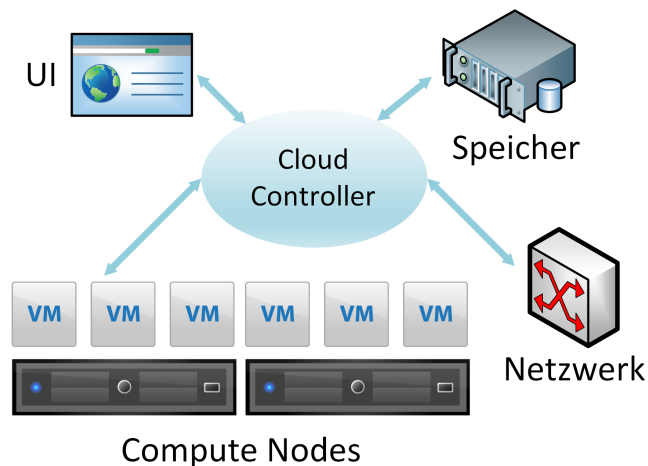


Abbildung 2.1: Generischer Aufbau von IaaS Clouds

In Abbildung 2.1 ist der generische Aufbau von IaaS Clouds aufgezeigt. IaaS Clouds sind im wesentlichen folgendermaßen aufgebaut: Als zentraler Verwaltungspunkt steht in der Mitte der sogenannte Cloud Controller. Dieser kümmert

sich um die Verwaltung von virtuellen Maschinen und deren Anbindung an die virtuelle und physische Netzwerkinfrastruktur. Dieser ist zuständig für das Zuweisen von Speicherplatz an die virtuellen Maschinen. Die virtuellen Maschinen selbst werden vom Cloud Controller auf sogenannte Compute Nodes verteilt. Dies sind physische Server auf denen ein Hypervisor installiert ist und auf dem dann die virtuellen Maschinen gestartet werden. Eine Benutzeroberfläche (meistens als Webinterface ausgelegt) bietet dann dem Endkunden das Verwalten seiner virtuellen Maschinen und Netzwerke an.

Einer der größten Vorreiter im Bereich IaaS ist Amazon Web Services mit ihrem Produkt EC2. EC2 hat sich durch seine große Nutzerbasis zum de facto Standard entwickelt was Schnittstellen nach außen hin angeht. Leider ist EC2 ein proprietäres Produkt wodurch es aus der Opensource Gemeinschaft ein großes Bestreben gab eine offene IaaS Softwarelösung zu schaffen. Im Laufe der letzten Jahre entstanden so einige Projekte wie OpenStack[2], Eucalyptus[4], oder OpenNebula[5] mit eben diesem Ziel. Weiterhin versuchen all diese Projekte die Schnittstelle von EC2 zu unterstützen, um so einen einfachen Umstieg zu ermöglichen.

2.2 OpenStack

OpenStack ist, wie im Kapitel zuvor, ein Softwareprojekt mit dem Ziel eine offene, kostenlose Infrastructure as a Service (IaaS) Cloud Platform zu schaffen. OpenStack wird von der OpenStack Foundation verwaltet, einer gemeinnützigen Organisation gegründet im September 2012. Die Entwicklung von OpenStack wird von vielen Firmen mitgetragen, unter anderem Rackspace, einer der führenden Anbieter von IaaS Cloudlösungen[6]. Die aktuelle¹ Version ist *Grizzly (2013.1.3)*. OpenStack legt viel Wert darauf die gleiche Schnittstelle wie Amazon Web Services anzubieten. Die OpenStack API ist kompatibel zu der von Amazon EC2 und Amazon S3 wodurch Werkzeuge und Programme mit minimalen Änderungen für beide Plattformen verwendet werden können. Diese Kompatibilität soll es Nutzern einfacher machen von Amazon Web Services zu OpenStack Anbietern zu wechseln, ohne dabei viel neues lernen zu müssen[7, 8].

2.2.1 Komponenten und Aufbau

OpenStack setzt sich aus mehreren Komponenten zusammen, welche im Folgenden aufgeführt werden. Einen Überblick über das Zusammenspiel der einzelnen Komponenten bietet Abbildung 2.2. *Keystone* ist der Identity Service von OpenStack. Er ist zuständig für die Authentifizierung von Benutzern gegenüber den anderen OpenStack-Diensten. Keystone kann dabei eine eigene Benutzerdatenbank verwenden oder auf bestehende zugreifen, zum Beispiel über LDAP.

¹Stand: 17.09.2013

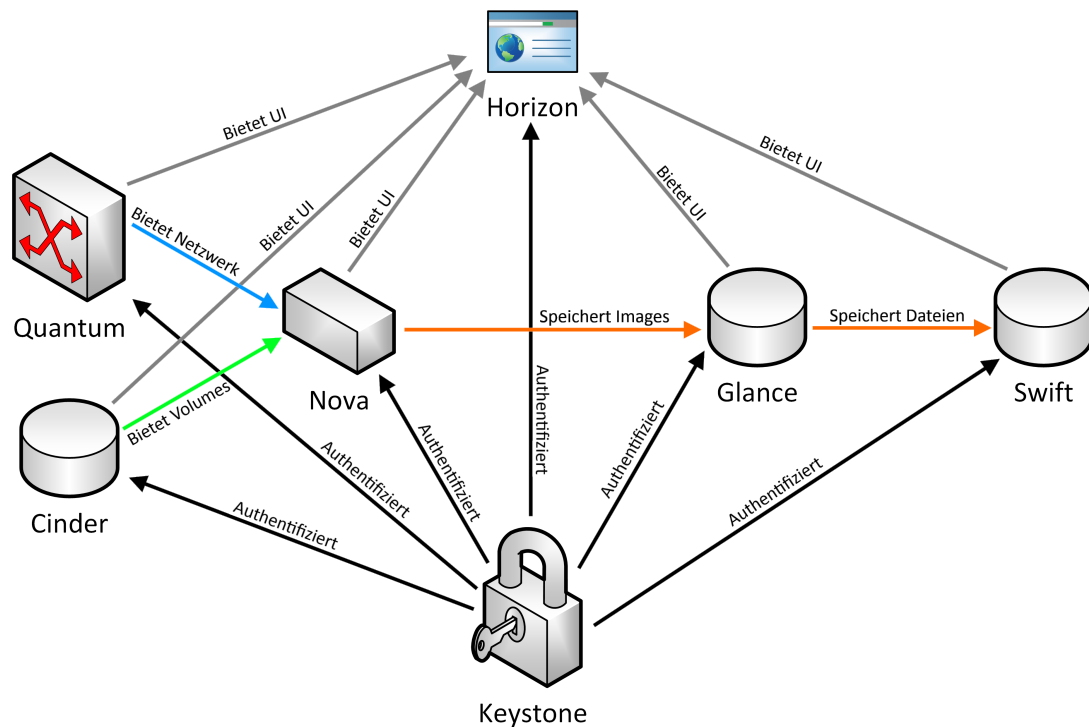


Abbildung 2.2: Aufbau und Zusammenspiel der OpenStack Komponenten

Swift ist der Object Storage von OpenStack. Er dient dazu Dateien redundant auf mehreren Servern zu speichern. Swift kümmert sich um die Replikation der Dateien und die Integrität des Clusters. Im Fehlerfall werden die betroffenen Dateien von anderen Servern weiterrepliziert. Da das Replizieren und Verteilen komplett in Software geschieht, ist Swift auch mit Hardware kompatibel, die nicht strengen Enterprise Spezifikationen entsprechen[9]. Desweiteren bildet Swift die Basis für *Glance*. Glance ist der Image Service von OpenStack und stellt Betriebssystem-Images bereit. Diese werden von Glance selbst in Swift abgelegt. Zur weiteren Verwendung werden die Images für *Nova* bereitgestellt. Nova (auch OpenStack Compute) ist der Cloud Controller (vgl. Abbildung 2.1) von OpenStack. Er ist der Hauptteil des OpenStack IaaS Systems und dient der Verwaltung von virtuellen Maschinen (VMs). Dazu kommuniziert Nova mit einem Hypervisor wie Xen um VMs zu starten/stoppen und diese zu verwalten. Nova kümmert sich ebenfalls um das Bereitstellen eines virtuellen Netzwerkes und um das Anbinden von Blockspeicher an die VMs. Snapshots von VMs werden von Nova erzeugt und dann als Image nach Glance hochgeladen.

Das virtuelle Netzwerk wird von *Quantum*² bereitgestellt. Quantum stellt IP-Adressen über das *Dynamic Host Configuration Protocol* (DHCP) für die VMs bereit und bietet einen *Domain Name System* (DNS) Server an. Weiterhin kann

²Wird in der nächsten Version in *Neutron* umbenannt

2 Basistechnologien

Quantum isolierte Netzwerke für einzelne VMs bilden. Da die Netzwerke standardmäßig von außerhalb nicht erreichbar sind, können mit *Floating IPs* IP-Adressen aus dem öffentlichen Raum einzelnen VMs zugewiesen werden. Dies erlaubt es nur einzelne Maschinen öffentlich erreichbar zu machen und andere, die nur intern erreichbar sein sollen, zu schützen. Die letzte Komponente heißt *Cinder* und stellt Nova, und damit den VMs, persistenten Blockspeicher bereit. Im Gegensatz zu Swift, welches seine Daten in Dateien ablegt, wird von Cinder direkter Hardwarezugriff auf Blockspeicher ermöglicht. Dies ist besonders für VMs von Vorteil, deren Leistung vom Datendurchsatz des Speichers abhängig ist. Zusätzlich unterstützt Cinder Snapshots für bereitgestellten Speicher. An die VMs angebunden wird der Blockspeicher über Nova.

All diese Komponenten werden von *Horizon* aus gesteuert. Horizon ist die Verwaltungsoberfläche von OpenStack in Form einer Webanwendung. Aufgebaut als modulare Django Webapp ist Horizon auch von außen aus erweiterbar. Django ist ein in Python geschriebenes Framework für dynamische Webseiten. Für Kernel as a Service ist Horizon die wichtigste Komponente. Über ein zusätzliches Modul wird Horizon um die Benutzeroberfläche von Kernel as a Service erweitert. Näheres dazu folgt in Kapitel [3.1](#).

2.3 Undertaker

Undertaker[3] ist ein Opensource Softwarepaket zum erleichterten Erstellen eigener Linux Kernel. Es wurde an der Universität Erlangen-Nürnberg entwickelt und im Paper *Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring*[1] im Februar 2013 vorgestellt. Undertaker entstand aus der Idee den Linux Kernel sicherer zu machen, indem nur benötigte Features im Kernel behalten werden und nicht benutzte Features ausgebaut werden.

Schaut man sich heutzutage Linux Distributionen und deren Kernel an, lässt sich feststellen, dass diese einen sehr großen Funktionsumfang besitzen. Dies hängt damit zusammen, dass die Anbieter der Linux Distributionen nicht wissen wofür und auf welcher Hardware ihre Distribution eingesetzt wird. Im Sinne der Kundenzufriedenheit wird daher die Kompatibilität erhöht, indem sehr viele Features in den Kernel eingebaut werden. Dies ist allerdings auch ein Nachteil, denn je mehr Features im Kernel vorhanden sind, desto größer ist die resultierende angreifbare Codebasis. Der Linux Kernel bietet allerdings eine große Vielfalt an Konfigurationsoptionen an, um nur die Features einzubauen, die gebraucht werden. Leider sind dies zur Zeit über 11000 verschiedene Optionen die sich teilweise ausschließen oder voneinander abhängen. Durch die Fülle an Optionen ist es nicht mehr trivial einen Kernel zu bauen, der exakt für den Anwendungsfall funktioniert.

Undertaker soll dabei helfen eine Konfiguration zu finden, die genau für den gewünschten Anwendungsfall passend ist. Dazu wird mit Hilfe des Kernelfeatures `ftrace` jeder Aufruf von Kernelfunktionen aufgezeichnet (im Folgenden als *tracen* bezeichnet). Mit solch einer Aufzeichnung lässt sich dann später eine Kernelkonfiguration errechnen, die nur noch die Features enthält, dessen Funktionen gebraucht wurden. Mit dieser Kernelkonfiguration lässt sich dann ein neuer Kernel erstellen (im Folgenden als *tailorn* bezeichnet).

Da undertaker noch recht neu ist, wird eine Benutzung nur auf Ubuntu 12.04 LTS (oder neuer) empfohlen, obwohl der theoretische Ansatz auch auf andere Distributionen anwendbar ist.

2.3.1 Bereitgestellte Werkzeuge und deren Benutzung

Undertaker stellt einige Werkzeuge zur Verfügung, jedoch sind für KaaS und den oben beschriebenen Anwendungsfall nur drei Werkzeuge relevant:

`undertaker-kconfigdump`, `undertaker-tracecontrol` und `undertaker-tailor`.

Wie diese Programme arbeiten und wie sie zusammen eingesetzt werden, wird im Folgenden erläutert.

Vorbereitungen

Bevor ein Kernel getraced werden kann, sind einige Voraussetzungen zu treffen. Der aktuelle Kernel muss mit `ftrace` Unterstützung gebaut worden sein, welches bei einem Ubuntu 12.04 bereits der Fall ist. Weiterhin muss der Quellcode des aktuellen Kernels heruntergeladen werden. Dabei ist darauf zu achten, die exakt gleiche Version herunterzuladen. Wenn zum Beispiel die aktuell laufende Version `3.2.0-45-virtual` ist und `apt-get source linux-image-3.2.0-45-virtual` die Version `3.2.0-53` herunterlädt, wird dies nicht funktionieren. Dieses Problem trat bei der Entwicklung von KaaS auf, siehe Kapitel 5.

Sind die Quellcode Dateien heruntergeladen, kann `undertaker-kconfigdump` eingesetzt werden. Dieses Programm erstellt nun zu einer gewählten Architektur ein sogenanntes *Model*. Dies geschieht indem der Quellcode untersucht wird und alle Codeblöcke die für die aktuelle Architektur nicht relevant sind ignoriert werden. Die übriggebliebenen, relevanten Codeblöcke werden nun weiter analysiert indem Funktionsnamen den Konfigurationsvariablen zugeordnet werden, die sie aktivieren. Die resultierenden Daten sind nun das Model des Kernels für eine Architektur. Dieser Schritt muss für jede gewünschte Architektur durchgeführt werden, allerdings nur ein einziges Mal pro Kernelversion.

Als letzte Vorbereitung müssen die Debugbinaries des Kernels heruntergeladen werden. Diese werden später benötigt, um Funktionsadressen in Funktionsnamen zu übersetzen.

Tracer starten

Als nächsten Schritt muss der Kerneltracer gestartet werden. Hier können zwei Varianten gewählt werden:

1. Start des Tracers so früh wie möglich während des Bootprozesses
2. Start des Tracers zu einem beliebigen Zeitpunkt nach dem Bootprozess

Von den `undertaker` Entwicklern wird die erste Variante empfohlen, da so große Teile des Bootprozesses mitgetraced werden und damit eher ein bootfähiger Kernel erzeugt werden kann, als mit der zweiten Variante. Um den Tracer schon während des Bootprozesses zu starten, ist eine modifizierte `initrd` nötig. Diese kann vollautomatisch von `undertaker-tracecontrol-prepare` erstellt werden.

Das System kann nach Installation der modifizierten `initrd` gestartet werden. Der Ablauf des Kerneltraces wird in Abbildung 2.3 dargestellt. Als erstes müssen alle Anwendungen, die getraced werden sollen, installiert und unter Last gesetzt werden. Im Hintergrund zeichnet das Kerneltool `ftrace` alle Funktionsaufrufe im Kernel auf. Ein Hintergrundprogramm von `undertaker` überwacht und kommuniziert mit `ftrace` und befüllt dynamisch dessen Blacklist mit den Funktionen die bereits aufgezeichnet worden sind. Weiterhin wird eine bereinigte Ausgabe in

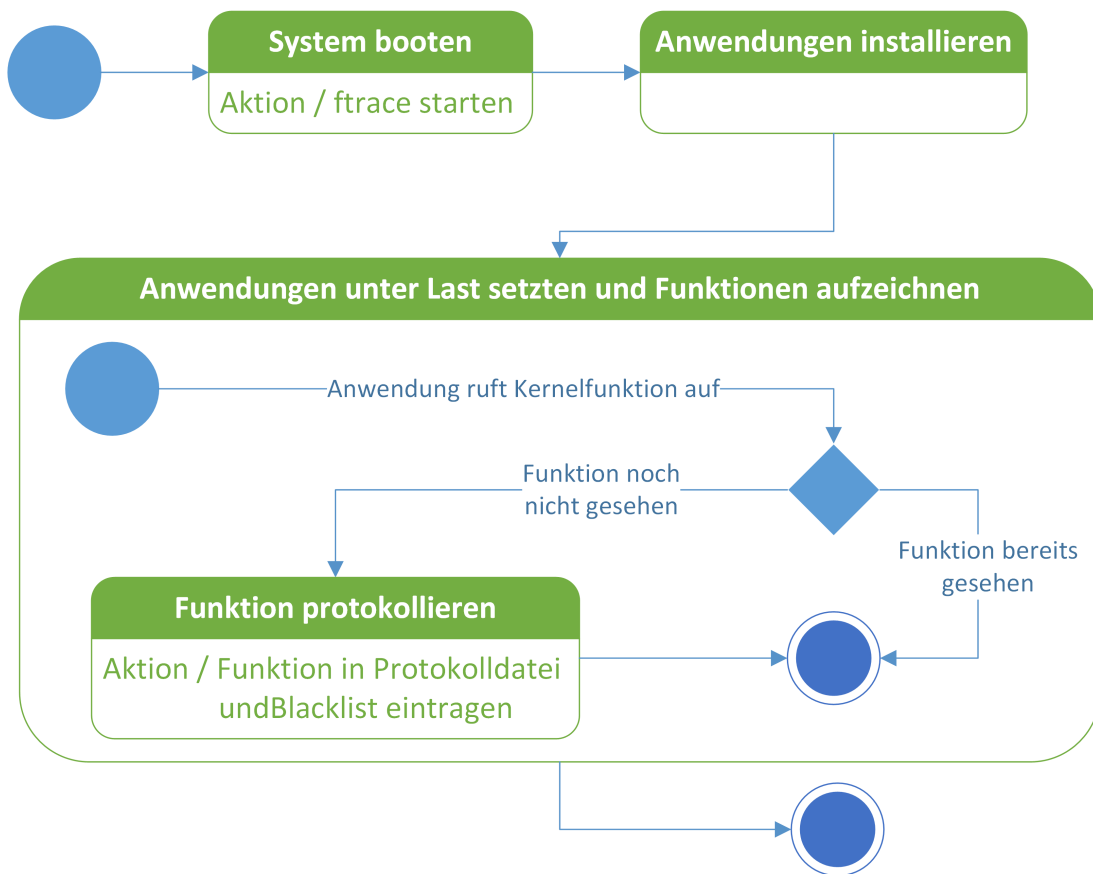


Abbildung 2.3: Zustandsautomat für das Tracen von Funktionen im Kernel

`/run/undertaker-trace.out` vorgenommen, in der nur noch die Funktionsadressen enthalten sind. Durch das erwähnte Blacklisting bereits gefundener Adressen enthält diese Datei keine Duplikate. `ftrace` ist allerdings nicht in der Lage alles aufzuzeichnen. So wird keine Interrupt-Behandlung, keine Kontextwechsel und `ftrace` selbst nicht aufgezeichnet.

Der Tracer sollte möglichst so lange laufen, wie neue Funktionen entdeckt werden. Das heißt der Trace ist abgeschlossen, wenn keine neuen Funktionen mehr von `ftrace` aufgezeichnet werden. Dieser Zeitpunkt ist allerdings nicht einfach bestimmbar. Die laufenden Anwendungen sollten hierfür so unter Last gesetzt werden, dass eine möglichst hohe Codeabdeckung gewährleistet ist, um so möglichst viele, benötigte Kernelfunktionen zu treffen. In der Praxis hat sich gezeigt, dass eine Wartezeit von mindestens einem Tag die untere Grenze sein sollte. Hier gilt, je länger desto besser.

Sollte es doch einmal zu dem Fall kommen, dass eine Funktion nicht getraced wurde, ist nicht gleich alles verloren. Beim Erstellen des Kernels werden nicht einzelne Funktionen ein- und ausgebaut, sondern Features. Features bestehen immer aus mehreren Funktionen. Wird eine Funktion nicht mitgetraced, die sich

in einem Feature befindet, das durch eine andere Funktion inkludiert wurde, ist auch die nicht getrace Funktion vorhanden. Problematisch wird es, wenn die nicht getrace Funktion in einem Feature liegt, welches nicht enthalten war. Nun hängt es von der Anwendung ab was passiert. Die Anwendung könnte prüfen, ob die Funktion vorhanden ist und entsprechende Maßnahmen ergreifen wenn nicht. Im schlimmsten Fall resultiert der Aufruf in einem Segmentation Fault und die Anwendung stürzt ab. Das Betriebssystem selbst stürzt nur ab, wenn für den Betrieb essentielle Funktionen fehlen. Diese werden aber normalerweise immer mitgetraced.

Der Tracer lässt sich mit dem Programm `undertaker-tracecontrol` beenden. Das gleiche Programm kann auch genutzt werden, um den Tracer zu einem beliebigen Zeitpunkt (neu-) zu starten. So können auch Tracepausen eingelegt werden. Dabei ist allerdings darauf zu achten, vorher die angelegte Ausgabedatei des vorherigen Traces zu sichern, da diese sonst überschrieben wird.

Erstellen einer Kernelkonfiguration

Mit einem vollständigen Trace lässt sich nun eine Kernelkonfiguration errechnen. Hierzu dient das Programm `undertaker-tailor`. Mit diesem Programm wird allerdings kein neuer Kernel gebaut, sondern lediglich eine valide Kernelkonfiguration errechnet. Der Tailor benötigt folgende Informationen um eine Kernelkonfiguration zu erstellen:

- Die bereinigte Ausgabe des Tracers (`/run/undertaker-trace.out`, die Protokolldatei)
- Das Model der Architektur der Protokolldatei
- Den Pfad zur Kernel Debugbinary
- Den Pfad zu den Kernelmodul Debugbinaries
- Eine White- und Blacklist für Kernelkonfigurationen
- Eine Ignorelist für Kernelquellcode Dateien

Das Maßschneidern des Kernels läuft folgendermaßen ab: Jede Funktionsadresse in der Protokolldatei wird mit Hilfe der Debugbinaries in den jeweiligen Funktionsnamen übersetzt. Funktionen, die in Quellcodedateien vorhanden sind, die auf der Ignorelist stehen, werden nicht beachtet. Mit den Funktionsnamen und dem Model wird nun eine Formel erstellt, dessen Variablen Codeblöcke, Konfigurationsoptionen oder ganze Quellcodedateien sind. Durch gezieltes Setzen von Konfigurationsoptionen wird nun eine Optionsmenge gesucht, durch die die Formel erfüllt werden kann. Die daraus resultierende Liste an Konfigurationsoptionen entspricht nun einer minimalen, vorläufigen Kernelkonfiguration. Nun werden die

White- und Blacklist relevant. Diese beiden Listen enthalten Konfigurationsoptionen, welche auf jeden Fall (Whitelist) bzw. auf keinen Fall (Blacklist) vorhanden sein müssen. Optionen, die noch nicht vorhanden sind, aber auf der Whitelist stehen, werden hinzugefügt. Optionen, die vorhanden sind, aber auf der Blacklist stehen, werden entfernt.

Aufgrund der komplexen Abhängigkeitsbeziehungen zwischen den Konfigurationsoptionen ist es wahrscheinlich, dass die entstandene Konfiguration noch keine gültige Konfiguration ist. Deshalb wird die vorläufige Konfiguration mit Hilfe von *Kconfig* zu einer validen Konfiguration erweitert.

Bauen des Kernels

Mit der fertigen Kernelkonfiguration lässt sich ein neuer Kernel bauen. Hierzu ist allerdings undertaker nicht mehr notwendig. Der Kernel kann zum Beispiel für Ubuntu direkt als installierbare `.deb` Datei mit `make deb-pkg` erstellt werden und dann mit `dpkg -i kernel.deb` installiert werden. Danach kann in den neuen Kernel gebootet werden. Überprüfen lässt sich die aktuelle Kernelversion im laufenden Betrieb mit `uname -r`.

3 Kernel as a Service

Kernel as a Service entstand aus der Idee, Nutzern von IaaS Clouds die Möglichkeit zu geben, die Sicherheit ihrer virtuellen Maschinen erheblich zu verbessern. IaaS Anbieter wie Amazon oder Rackspace bieten ihren Kunden Templates von virtuellen Maschinen an, die oft mit einem generischen Linux Kernel arbeiten. Diese generischen Kernel enthalten aus Kompatibilitätsgründen viele Features, welche häufig nicht benötigt werden. Selten oder gar nicht verwendete Features können Sicherheitslücken enthalten, durch die ein Angreifer Zugriff auf das eigene System gewinnen kann. Im Interesse der Sicherheit ist von Vorteil, wenn diese Features gar nicht erst im Kernel vorhanden sind. Leider ist es durch eine Fülle an verfügbaren Konfigurationsmöglichkeiten nicht einfach einen maßgeschneiderten Kernel von Hand zu erstellen. Das in Kapitel 2.3 vorgestellte Programmpaket *undertaker* möchte Benutzern dabei helfen, die richtigen Konfigurationsoptionen auszuwählen. Leider ist der Prozess des Kernelbauens immernoch manuell zu erledigen.

KaaS soll nun dabei helfen, einen maßgeschneiderten Kernel zu erstellen, indem der gesamte Prozess des Kernelbauens automatisiert wird. Weiterhin soll es dem Benutzer sehr einfach gemacht werden einen neuen Kernel zu erstellen.

Kernel as a Service wurde während der Designphase als dreiteiliges System entworfen:

1. Das KaaS Horizon Modul
2. Der KaaS Buildserver
3. Die KaaS Nutzer VM

Der grobe Aufbau dieser drei Teile wird im Folgenden erläutert. Eine Erklärung der Implementierung folgt in Abschnitt 4.

3.1 KaaS Horizon Modul

Wie bereits in Kapitel 2.2.1 erwähnt, wird die Benutzeroberfläche von OpenStack von *Horizon* bereitgestellt. Horizon ist eine Django Webapp und damit in Python geschrieben. Die anderen OpenStack Komponenten werden von Horizon aus Verwaltet und gesteuert. Die gesamte Oberfläche wird auch als *Dashboard*

3 Kernel as a Service

bezeichnet. Das Dashboard setzt sich aus drei¹ Modulen zusammen: *Admin*, *Project* und *Settings*. KaaS soll als viertes Modul hinzugefügt werden, ohne dabei die anderen Module zu verändern.

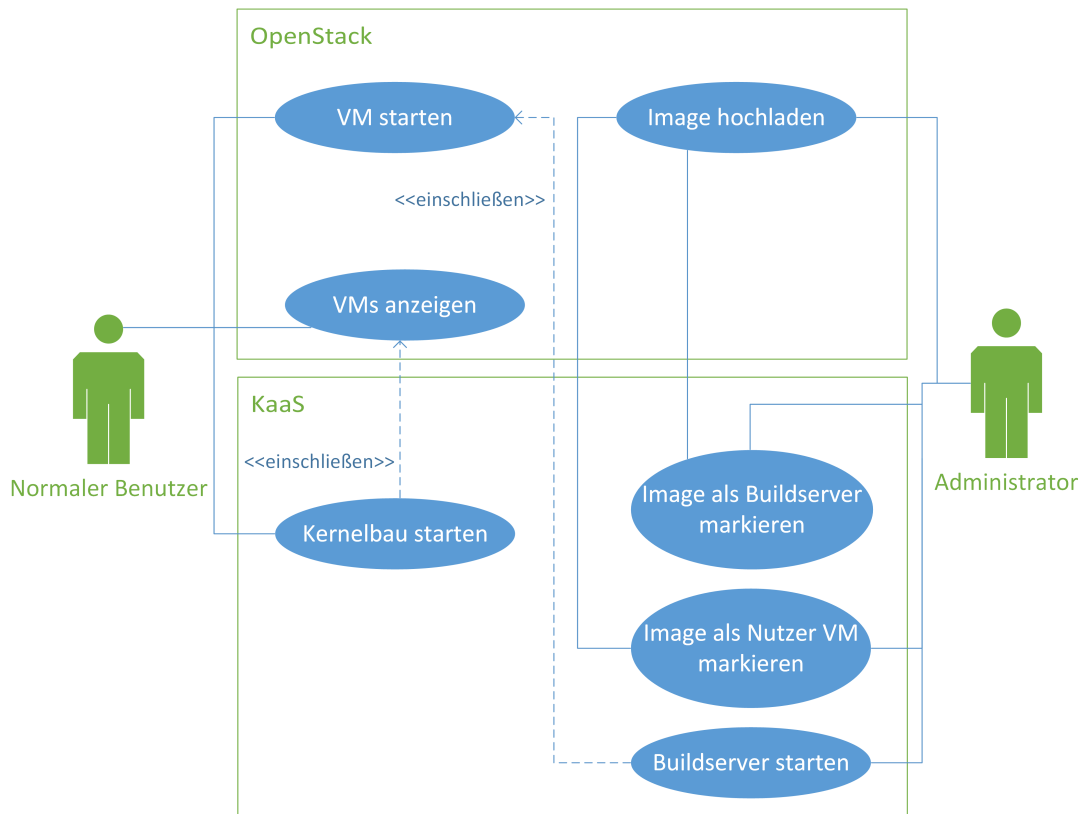


Abbildung 3.1: Anwendungsfallsdiagramm für die beiden Nutzertypen von KaaS. Für OpenStack wurden nicht alle möglichen Anwendungsfälle angegeben.

Das KaaS Horizon Modul soll für Administratoren und Nutzer unterschiedliche Funktionalitäten anbieten. Abbildung 3.1 zeigt die verschiedenen Anwendungsfälle der beiden Nutzertypen.

Für die OpenStack Nutzer soll die Möglichkeit gegeben werden, Kernel bauen zu können. Dazu soll, wie im Project Modul, eine Liste der gestarteten virtuellen Maschinen angezeigt werden mit einer Schaltfläche zum Starten eines Kernelbaus. Hierbei dürfen nur VMs angezeigt werden, die auch eine KaaS Nutzer VM (und damit tracebar) sind. Die Unterschiede zwischen einer KaaS Nutzer VM und einer normalen Nutzer VM werden in Abschnitt 3.3 beschrieben.

Für OpenStack Administratoren wird zusätzlich die Einstellungsseite des KaaS Moduls erreichbar. Hier ist einzustellen, welche Images in Glance KaaS Nutzer

¹Stand: 17.09.2013

VMs sind, und welche Images KaaS Buildserver sind. Hier sind auch die KaaS Buildserver zu starten, welche im folgenden Abschnitt erläutert werden.

3.2 KaaS Buildserver

Der KaaS Buildserver ist ein speziell angepasstes VM Image. Der Buildserver ist für das Erstellen der Kernel zuständig und muss daher eine Installation von undertaker beinhalten. Weiterhin muss eine Möglichkeit zur Kommunikation mit dem Horizon Modul und der Nutzer VM geschaffen werden. Hierzu ist ein Programm vorgesehen, welches auf eingehende Befehle wartet und diese dann ausführt. Der Buildserver muss nur einen Befehl entgegennehmen, nämlich den des Kernelbauens. Informationen zu dem Kernelbau werden als Argumente mitgegeben. Somit verhält sich das System wie bei einem entfernten Methodenaufruf. Weitere Befehle, wie zum Beispiel das Abfragen des aktuellen Status, sind optional aber wünschenswert.

Das Buildserver VM Image dient nicht dazu an Nutzer gegeben zu werden, sondern soll von den Administratoren über das Horizon Modul gestartet werden (siehe Abschnitt 3.1). Eine so gestartete Buildserver VM steht damit allen Nutzern zur Verfügung, die einen Kernelbau anfordern.

3.3 KaaS Nutzer VM

Wie der Buildserver ist auch die KaaS Nutzer VM ein speziell angepasstes VM Image. Im Hintergrund muss die Nutzung des Kernels aufgezeichnet werden. Somit ist ebenfalls eine undertaker Installation notwendig. Wie auch beim Buildserver muss mit dieser VM kommuniziert werden können. Hier kommt auch wieder ein Programm zum Einsatz. Da dies aber kein Buildserver ist, akzeptiert dieses Programm andere Befehle. Ein Befehl wird für das Stoppen des Kerneltracers verwendet, ein zweiter zum Abrufen der Protokolldatei. Der dritte Befehl dient zum Abrufen des Zeitpunktes der zuletzt gefundenen, neuen Kernelfunktion. Dieser Befehl ist notwendig um dem Benutzer im Horizon Modul zu informieren, ob er einen Kernelbau starten oder lieber noch warten sollte.

KaaS geht davon aus, dass allen Benutzern die gleiche Nutzer VM bereitgestellt wird, also alle den gleichen Ursprungs Kernel benutzen. Die Buildserver VMs können damit auch nur für exakt diesen Kernel verwendet werden. Es wäre allerdings möglich mehrere Nutzer VM Images mit unterschiedlicher vorinstallierter Software bereitzustellen. Es muss nur der verwendete Kernel zwischen allen VM Images gleich sein.

4 Aufbau und Implementierung

Die finale Implementierung von Kernel as a Service entspricht größtenteils dem beschriebenen Design aus Kapitel 3. Das KaaS Horizon Modul wurde zusätzlich um eine simple Verwaltung der eigenen Images erweitert. Diese Verwaltung erlaubt das Umbenennen und Löschen der von KaaS erstellten Images. In den nächsten Kapiteln folgt nun die Erläuterung der fertigen Implementierung.

4.1 Gemeinsam verwendeter Basis Server

Für den KaaS Buildserver und die KaaS nutzer VM wird ein Programm benötigt, welches Befehle annimmt, diese ausführt und gegebenenfalls ein Ergebniss zurückschickt. Dazu wurde eine Basis entwickelt auf der die beiden benötigten Programme aufbauen. Diese Basis wurde wie das KaaS Horizon Modul in Python entwickelt und besteht aus zwei Klassen:

- Die Klasse `Connection` zum Herstellen von Socketverbindungen und zum Verschicken und Empfangen von beliebigen Nachrichten.
- Die Klasse `workerThread` zum Abarbeiten eingehender Befehle.

Zusätzlich vorhanden ist die `main`-Methode zum Starten des Listener Threads welche keiner Klasse angehört.

Die Klasse `Connection` benutzt das Python Modul `socket` um Socketverbindungen zu erstellen. Jede Instanz der Klasse steht dabei für eine hergestellte Verbindung, bzw. für eine Socketverbindung. Die Klasse bietet nur die notwendigsten Funktionen: Mit `listenOn` und `connectTo` werden Socketverbindungen zwischen zwei Instanzen hergestellt. Über die Methoden `send` und `receive` können dann Nachrichten verschickt und empfangen werden. `close` schließt die geöffnete Verbindung.

Von der `main`-Methode wird auf Port 1030 ein Serversocket erstellt, welcher auf eingehende Verbindungen wartet. Port 1500 wurde gewählt, da dieser Port offiziell keinem Programm zugeordnet wurde[10]. Weiterhin ist 1030 größer als 1024 und es werden somit keine root-Rechte auf UNIX-artigen Betriebssystemen benötigt um einen Serversocket zu öffnen.

Eingehende Verbindungen werden von der Klasse `workerThread` behandelt. Für jede Verbindung wird eine Instanz dieser Klasse erzeugt, welche in einem eigenen Thread eingehende Befehle abarbeitet. Um Shellbefehle auszuführen wird

4 Aufbau und Implementierung

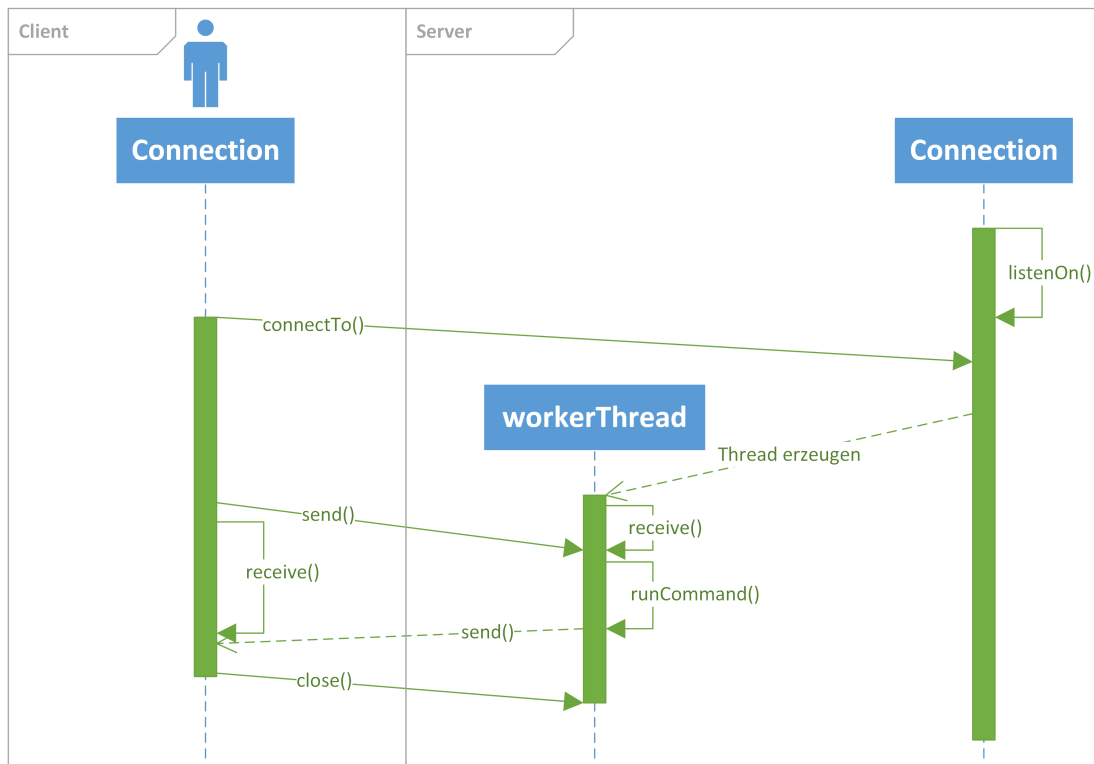


Abbildung 4.1: Sequenzdiagramm für eine Client-Server Verbindung mit der `Connection` Klasse

die Methode `runCommand` bereitgestellt, welche dies mit Hilfe des Python Moduls `subprocess` tut. Der Ablauf des Kernelbauens wird dadurch stark vereinfacht, da das Python Programm so die gleichen Befehle ausführt wie ein Mensch in einer Shell. Abbildung 4.1 zeigt den Ablauf einer Verbindung mit der `Connection` Klasse. Gut zu erkennen ist, dass jede Verbindung ihren eigenen Thread bekommt welcher nur so lange läuft, wie die Verbindung aktiv ist.

4.2 KaaS Buildserver

Der KaaS Buildserver basiert auf einem Ubuntu 12.04 Cloud Daily Build Image¹ vom 11. Juni 2013. Die Cloud Daily Build Images wurde gewählt, da diese bereits für die Benutzung in einer IaaS Cloud Umgebung angepasst sind. Installiert wurde `undertaker` in der Version 1.4. Mit dem Buildserver startet auch ein Python Programm, welches sich um die in Abschnitt 3.2 beschriebenen Befehle kümmert (genannt *KaaS Python Buildserver*). Es basiert auf dem in Abschnitt 4.1 beschriebenen Programm und ist um die Funktionalität des Kernelbauens erweitert. Die Buildserver Variante nimmt die Befehle `startbuild`, `status` und `buildstatus`

¹Verfügbar unter <http://cloud-images.ubuntu.com/precise/current/>

an.

Der Befehl `status` fragt den aktuellen Status des Buildservers ab und wird nur von der Administrationsseite des KaaS Horizon Moduls gesendet. Ist der Server gerade im Idle, wird `free` zurückgegeben. Ist der Server gerade dabei einen Kernel zu bauen, wird der aktuelle Status des laufenden Kernelbaus und die Länge der Warteschlange zurückgegeben. Die Warteschlangenlänge beträgt dabei immer mindestens eins, da der laufende Kernelbuild auch dazugezählt wird. Das KaaS Horizon Modul muss daher vor der Anzeige der Warteschlangenlänge eins subtrahieren.

Um Informationen über einen bestimmten Kernelbuild anzufragen, wird der Befehl `buildstatus` verwendet. Hierzu muss dem Buildserver nach dem Befehl noch die Id der VM mitgeteilt werden, dessen Status abgefragt werden soll. Dazu sendet der Buildserver nach Empfang des Befehls ein `go` zurück um die Bereitschaft zur Annahme der Id zu signalisieren. Das KaaS Horizon Modul sendet dann die Id der VM und enthält darauf den aktuellen Status des Kernelbaus.

Der aktuelle Status eines Kernelbaus wird durch eine Zahl zwischen -1 und 6 repräsentiert. Die Bedeutung der einzelnen Statuscodes folgt in Tabelle 4.1.

Tabelle 4.1: Statuscode eines Kernelbaus und deren Bedeutung

Statuscode	Bedeutung
-1	Fehler beim Kernelbau
0	Noch nicht gestarteter Auftrag (noch in der Warteschlange)
1	Abrufen der Protokolldatei von der Nutzer VM
2	Erstellen der Kernelkonfiguration durch undertaker
3	Bauen des Kernels
4	Installieren des Kernels
5	Hochladen des fertigen Images
6	Abgeschlossener Kernelbau

Zum Starten eines Kernelbaus wird der Befehl `startbuild` verwendet. Der Buildserver muss dazu die IP-Adresse, die Id und den Besitzer der VM, für die ein Kernelbau angefordert wurde, kennen. Dazu wird nach Empfang des Befehls wie beim `buildstatus` Befehl ein `go` versendet, sobald der Buildserver bereit ist die Informationen entgegenzunehmen. Das KaaS Horizon Modul sendet dann die benötigten Informationen in folgender Reihenfolge: IP-Adresse, Besitzer Id, VM Id. Ein neues Kernelbauobjekt wird erstellt und in die Warteschlange eingereiht. Falls es bisher keinen anderen laufenden Kernelbau gibt, wird ein neuer Kernelbauthread mit den Informationen gestartet. Zu allerletzt wird dem KaaS Horizon Modul mitgeteilt, dass der Kernelbau beginnen kann.

4 Aufbau und Implementierung

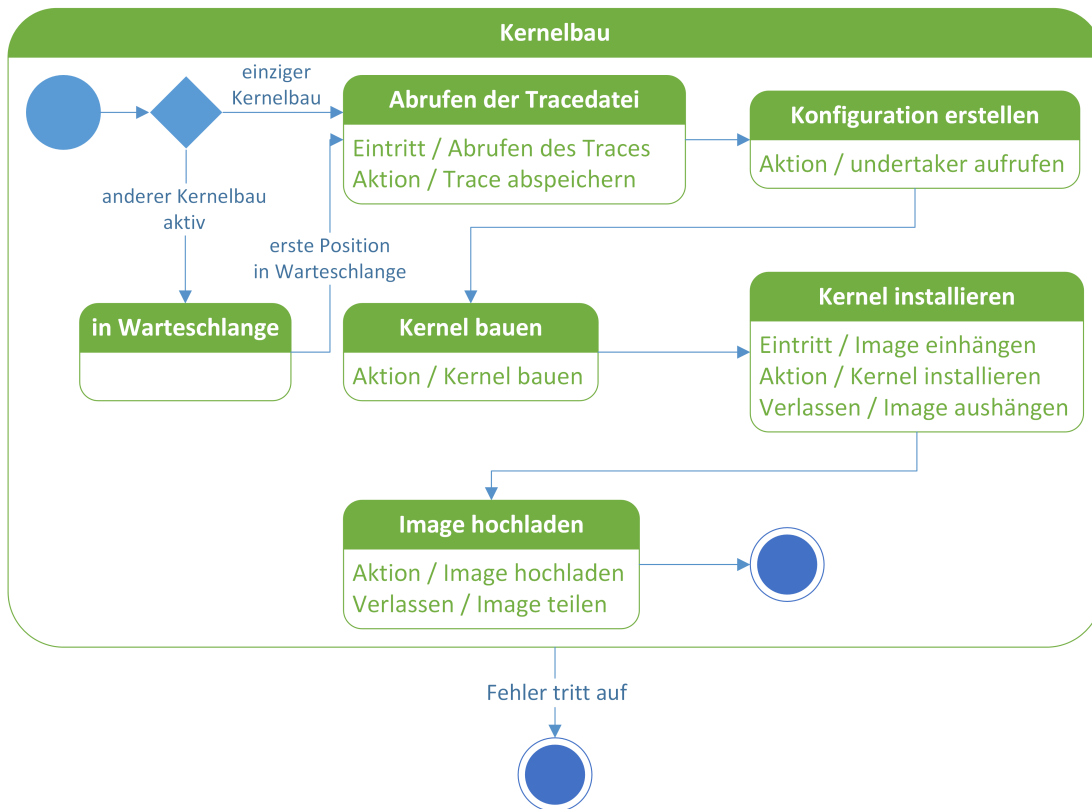


Abbildung 4.2: Zustandsautomat für einen Kernelbau

4.2.1 Ablauf des Kernelbaus

Ein Kernelbau folgt immer dem gleichen Schema wie in Abbildung 4.2 zu sehen ist. Die Zustände korrespondieren zu den gleichnamigen Statuswerten aus dem vorherigen Abschnitt.

Der Buildserver muss zuerst die Protokolldatei abrufen. Dazu verbindet er sich zur angegebenen IP-Adresse und sendet den Befehl `gettrace`, welcher in Abschnitt 4.3 beschrieben wird. Die erhaltene Protokolldatei wird durch die VM Id identifizierbar abgespeichert. Der Kernelbau startet nun mit dem Erstellen einer Kernelkonfiguration wie in Kapitel 2.3.1 beschrieben. Der Buildserver führt die notwendigen Befehle aus und orientiert sich dabei am *Tailor How To*². Der fertige Kernel muss nun installiert werden. Dazu liegt im Buildserver ein Ubuntu 12.04 Image vor. Dieses wird kopiert und die Kopie wird nach `/mnt` eingebunden. Der Kernel wird dann installiert, indem das erzeugte Debian-Paket entpackt wird und die GRUB Konfiguration auf den neuen Kernel angepasst wird. Der Kernel kann leider nicht direkt mittels `dpkg` in einer `chroot` Umgebung installiert werden. Die Installationsskripte können aufgrund des `chroot` nicht die GRUB

²Verfügbar unter <http://vamos.informatik.uni-erlangen.de/trac/undertaker/wiki/TailorHowto>

Konfiguration korrekt anpassen. Siehe dazu Kapitel 5.

Das fertige Image wird nun zu Glance hochgeladen und mit dem Besitzer der originalen VM geteilt. Abschließend wird das Image vom Buildserver gelöscht. Dieser Kernelbau ist hiermit abgeschlossen und es kann der nächste Kernelbau gestartet werden. Dies geschieht bei nicht leerer Warteschlange automatisch.

4.3 KaaS Nutzer VM

Die KaaS Nutzer VM basiert auf dem gleichen Ubuntu 12.04 Cloud Daily Build wie das KaaS Buildserver VM Image. Auch hier wird ein Python Programm beim Boot gestartet, welches sich um die in Abschnitt 3.3 beschriebenen Befehle kümmert (genannt *KaaS Python Trace Server*). Wie auch das Programm in der Buildserver VM basiert dieses auf dem in Abschnitt 4.1 beschriebenen Python Basis Server, ist allerdings weniger komplex. Der Trace Server nimmt drei Befehle entgegen: `stoptracing`, `gettrace` und `getlastfunctiontime`.

Der Befehl `stoptracing` stoppt den laufenden Tracer. Die Protkolldatei bleibt dabei unberührt. Diese kann mit dem Befehl `gettrace` abgerufen werden. Damit das KaaS Horizon Modul seine Nutzer informieren kann ob ein Kernelbau sinnvoll ist oder noch weiter getraced werden sollte, ist der Befehl `getlastfunctiontime` notwendig. Er gibt den Zeitpunkt der Entdeckung der zuletzt getraceten, noch unbekannt Kernelfunktion zurück. Dies wird bewerkstelligt, indem die Größe der Protkolldatei überwacht wird. Beim ersten Aufruf der Funktion wird die Größe der Protkolldatei gespeichert. Bei jedem weiteren Aufruf wird die gespeicherte Größe mit der neu ausgelesenen verglichen. Ist die Größe gleich, wurden keine neuen Kernelfunktionen entdeckt. Ist die Datei größer als vorher, wurden neue Funktionen entdeckt und deren Adressen in die Protkolldatei geschrieben. Nun merkt sich die Funktion den Zeitpunkt der letzten Modifikation der Datei. Dieser Zeitpunkt wird dann zurückgegeben. Codelisting 4.1 zeigt die Implementierung dieser Funktion in Python.

4.4 Eigenbau der VM Images

Anstelle der Referenzimages ist es auch möglich die VM Images selbst zu erstellen. Als Basis sollte dabei immer ein Ubuntu 12.04 genommen werden, da dies von den undertaker Entwicklern empfohlen wird. Im nächsten Schritt muss undertaker installiert werden. KaaS verwendet die aktuelle Version 1.4³. Zusätzlich müssen im Buildserver VM Image die Kernel sources und die Kernel Debugbinaries für den Kernel der Nutzer VM heruntergeladen, bzw. installiert werden. Weiterhin muss auch der KaaS Python Buildserver installiert und konfiguriert werden. Eine Kopie des verwendeten Basis Images muss auch vorhanden sein, da es als Basis

³Stand: 04.10.2013

4 Aufbau und Implementierung

```
1 import os.path
2 import datetime
3 from threading import Lock
4
5 currentsize = 0
6 time = datetime.datetime.now()
7 timelock = Lock()
8
9 def getlastfunctiontime(self):
10     timelock.acquire()
11     t = os.path.getmtime("/run/undertaker-trace.out")
12     csize = os.path.getsize("/run/undertaker-trace.out")
13     if csize > currentsize:
14         currentsize = csize
15         time = datetime.datetime.fromtimestamp(t)
16     timelock.release()
17     return time
```

Listing 4.1: Python Funktion zum Ermitteln des Zeitpunktes der zuletzt getraceten Funktion

für die maßgeschneiderten Kernel dient. Es ist daher wahrscheinlich notwendig das Buildserver Image zu vergrößern um genug Platz zu haben.

Für das Nutzer VM Image muss von undertaker eine modifizierte *initrd* erstellt werden um den Tracer beim Booten zu starten. Weiterhin muss der KaaS Python Trace Server installiert werden. Eine genaue Schritt-für-Schritt Anleitung liegt dem KaaS Horizon Modul bei.

4.5 KaaS Horizon Modul

Das KaaS Horizon Modul wird, wie in Abschnitt 3.1 beschrieben, als Modul ausgeführt um einfach zu den bestehenden Horizon Modulen hinzugefügt werden zu können. Das Horizon Modul besteht aus drei Teilen:

- Eine simple Einstellungsdatei
- Nur für Administratoren sichtbarer Verwaltungsbereich
- Für alle sichtbarer Übersichtsbereich

Einstellungsdatei

KaaS speichert selbst nur wenige Daten weswegen auf eine Einstellungsdatei, statt einer Datenbank, zurückgegriffen wird. Diese Datei enthält ein simples JSON-Objekt welches alle notwendigen Informationen enthält um KaaS zum Laufen

zu bringen. Weiterhin wird dieses Objekt auch benutzt um im Webinterface vorgenommene Einstellungen abzuspeichern. Ein Beispiel für eine valide KaaS Einstellungsdatei folgt in Codelisting 4.2.

```

1 {
2     "kaasuser": {
3         "username": "kaas",
4         "authurl": "http://10.0.0.1:5000/v2.0",
5         "password": "kaaspassword",
6         "project": "kaas_upload",
7         "tenant" : "e25a536f30a14d16a46340d15929da65"
8     },
9     "traceableimages": [],
10    "buildserverimages": [],
11    "builds": {}
12 }
```

Listing 4.2: Beispiel einer validen KaaS Einstellungsdatei

KaaS benötigt einen eigenen Benutzeraccount und Projekt in OpenStack (siehe Abschnitt 4.6). Die Informationen über diesen Account werden unter dem Eintrag *kassuser* gespeichert. Dies sind auch die einzigen Einstellungen, die von Hand in der Datei vorgenommen werden müssen. Die Listen *traceableimages* und *buildserverimages* und die Map *builds* werden von KaaS gefüllt, nachdem im Verwaltungsbereich Images zugeordnet, bzw. Kernel gebaut worden sind (siehe Abschnitt 4.5).

Übersichtsbereich

Dieser Übersichtsbereich wird jeden OpenStack Benutzer angezeigt. Abbildung 4.3 zeigt wie diese Seite für einen Benutzer aussieht.

Die obere Tabelle zeigt alle vom Nutzer gestarteten Instanzen mit installiertem Tracer, also alle gestarteten KaaS Nutzer VMs. Diese Liste wird erstellt, indem alle VMs des Nutzers abgefragt werden und alle VMs rausgefiltert werden, dessen Images keiner KaaS Nutzer VM entsprechen. Dazu muss von einem Administrator vorher ein Image als KaaS Nutzer VM markiert werden (siehe Abschnitt 4.5). Ein Klick auf den Namen der VM führt zur Detailseite, welche vom Horizon Projects Modul bereitgestellt wird. Weiterhin bietet die Tabelle dem Nutzer Informationen über die IP Adressen der VMs sowie den Status eventuell laufender Kernelbauanfragen. Die IP Adresse wird mit aufgeführt um VMs unterscheiden zu können, die den gleichen Namen besitzen. Letztendlich lässt sich durch einen Klick auf *Build a Kernel* eine Kernelbauanfrage absenden. Die Kernelbauanfrage wird allerdings nicht sofort abgesendet. Der Benutzer wird zuerst auf eine Zwischenseite, wie in Abbildung 4.4 zu sehen, weitergeleitet.

4 Aufbau und Implementierung

Overview Logged in as: demo [Settings](#) [Help](#) [Sign Out](#)

Traceable Instances

Instance Name	IP Address	Status	Actions
u1	10.0.0.3	Done	Build a new kernel

Displaying 1 item

Images with tailored Kernels

[Delete Images](#)

<input type="checkbox"/>	Image Name	Actions
<input type="checkbox"/>	KaaS build for u1	Launch More

Displaying 1 item

Abbildung 4.3: Übersichtsbereich des KaaS Horizon Moduls. Sichtbar für alle Benutzer

Build a Kernel

Build a Kernel

Let me build anyway *

Continue tracing

KaaS recommendation
The last, previously unknown function was traced on 20.10.2013 10:53:16 UTC.
This was 3 hour(s) 46 minute(s) and 40 second(s) ago.
There might be additional functions being traced in the future. You should wait with your kernel build.

[Cancel](#) [Build a Kernel](#)

Abbildung 4.4: Zwischenseite auf dem Weg zum Kernelbau. KaaS empfiehlt mit dem Kernelbau zu warten

KaaS informiert hier den Nutzer hier über den Zeitpunkt der zuletzt getrackten Funktion. Sollte dieser Zeitpunkt nicht länger als einen Tag zurückliegen, wird von KaaS die Empfehlung ausgesprochen noch keinen Kernel zu bauen, da noch weitere Funktionen hinzukommen könnten. Andernfalls wird die Empfehlung ge-

geben, dass ein Kernelbau in Ordnung ist. Jenachdem wird auch die *Let me build anyway* Checkbox angezeigt. Im ersten Fall, also Empfehlung zu warten, ist die Checkbox sichtbar. Dem Benutzer wird das Absenden einer Kernelbauanfrage solange verwehrt, wie diese Checkbox nicht ausgewählt ist. Somit kann kein versehentlicher Kernelbau angestoßen werden. Im zweiten Fall, also Empfehlung nicht zu warten, ist diese Checkbox nicht vorhanden (siehe Abbildung 4.5). Die zweite Checkbox erlaubt es dem Benutzer KaaS mitzuteilen, dass der Tracer nicht beendet werden soll. Dies erlaubt es einen Kernel zu bauen und trotzdem weiterhin zu tracen. Dies ist sinnvoll, wenn zum Beispiel noch nicht feststeht, ob noch weitere Programme installiert werden müssen und somit weitere Funktionen hinzukommen könnten. Letztendlich kann mit dem *Build a Kernel* Knopf die Kernelbauanfrage versendet (siehe Abschnitt 4.2) oder mit *Cancel* abgebrochen werden.

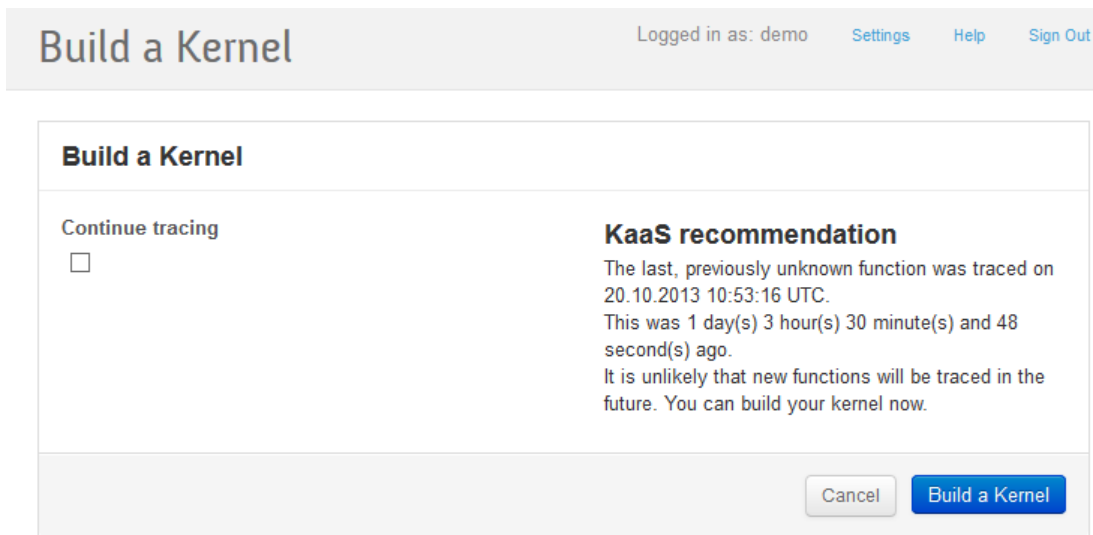


Abbildung 4.5: Zwischenseite auf dem Weg zum Kernelbau. KaaS empfiehlt, dass mit dem Kernelbau begonnen werden kann

In der unteren Tabelle auf der Übersichtsseite werden alle Images aufgelistet, die aus Kernelbauanfragen des Benutzers entstanden sind. Die Images mit maßgeschneidertem Kernel können hier direkt gestartet werden. Weiterhin wird dem Benutzer angeboten das Image umzubenennen oder zu löschen. Diese Funktionen können leider nicht aus der normalen Imageverwaltung durchgeführt werden, da die Images nicht dem Benutzer direkt gehören, sondern einem KaaS eigenen Projekt (siehe dazu Abschnitt 4.6).

Administrativer Verwaltungsbereich

Im administrativen Verwaltungsbereich können von den OpenStack Administratoren diverse Einstellungen zu KaaS vorgenommen werden. Wie in Abbildung 4.6

4 Aufbau und Implementierung

zu sehen können hier die Images ausgewählt werden, welche von KaaS als tracebar angesehen werden und welche als Buildserver benutzt werden. Die Buildserver werden auch hier gestartet und beendet.

The screenshot shows the 'Settings' page for an administrator. At the top right, it says 'Logged in as: admin' with links for 'Settings', 'Help', and 'Sign Out'. The page is divided into three sections:

- Traceable Images:** Contains a table with one entry: 'Traceable Ubuntu 12.04' with image ID 'f1544826-0f5b-4d17-9708-8f0dce37b32e'. A 'Delete Image' button is present.
- Buildserver Images:** Contains a table with one entry: 'KaaS Buildserver 4' with image ID '54414a85-9db2-4d60-abd6-04e097c4ab84'. A 'Delete Image' button is present.
- Buildserver VMs:** Contains a table with one entry: VM ID 'bcd216e7-8e14-4a38-8a63-e1ad86fa7852', size 'm1.medium', and status 'free'. A 'Terminate Instance' button is present.

Each section has '+ Add' and 'Delete' buttons. The status 'Displaying 1 item' is shown at the bottom of each table.

Abbildung 4.6: Administrativer Verwaltungsbereich des KaaS Horizon Moduls. Sichtbar nur für Administratoren

In der obersten Tabelle werden alle Images angezeigt, die als *tracebar* markiert wurden, also alle KaaS Nutzer VM Images. Die Images werden ganz normal zu OpenStack hochgeladen und dann hier, wie in Abbildung 4.7 zu sehen, als tracebar markiert. Intern wird von KaaS dann die Image ID in der Einstellungsdatei abgespeichert. Dieses Zuordnen ist notwendig um den Benutzern im Übersichts-bereich nur Images anzuzeigen, die auch KaaS Nutzer VMs sind.

In der mittleren Tabelle werden analog die Buildserver Images eingestellt. Diese Images werden dann benutzt um Buildserver VMs zu starten. Die in beiden Tabellen vorhandenen *Delete Image* Knöpfe entfernen die Zuordnung wieder, sie löschen das Image aber nicht.

In der untersten Tabelle werden alle gestarteten Buildserver aufgelistet, also

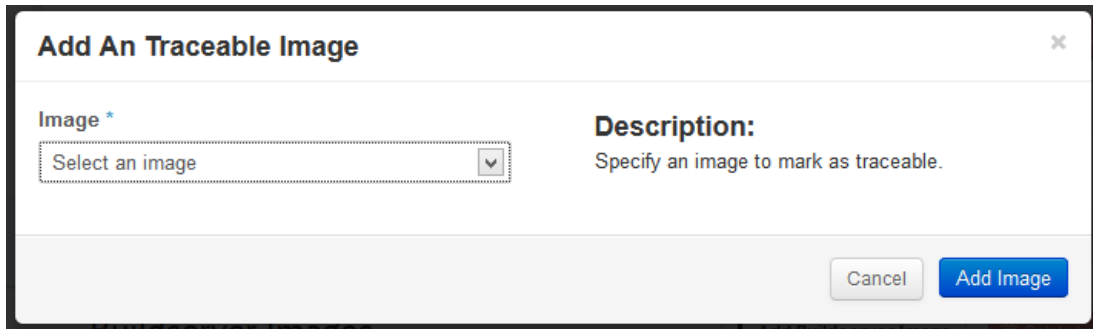


Abbildung 4.7: Formular zum Einstellen welche Images KaaS Nutzer VMs sind

alle laufenden VMs, dessen Images Buildserverimages sind. In der Tabelle wird auch der aktuelle Status des Buildservers angezeigt. Zusätzlich zum Status des aktuell laufenden Kernelbaus wird auch die Länge der Warteschlange auf jedem Buildserver angezeigt. Die Buildserver VMs werden auch von hier aus gestartet (siehe Abbildung 4.8). Es ist nicht möglich die Buildserver VMs aus dem Project Modul zu starten, da KaaS die Benutzerdaten des KaaS Benutzers (siehe Abschnitt 4.6) der Buildserver VM beim Start mit übergibt.

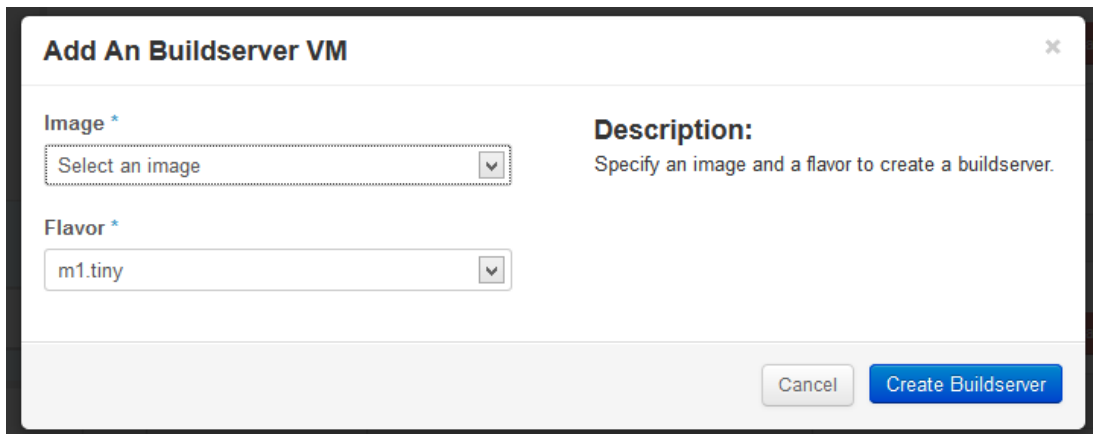


Abbildung 4.8: Formular zum Starten von KaaS Buildservern

4.6 Zusammenspiel der Komponenten

Um KaaS in OpenStack zu integrieren reicht es nicht nur das Horizon Modul hinzuzufügen. Weitere Vorarbeiten müssen geleistet werden welche im Folgenden erläutert werden.

Innerhalb der Horizon WebApp werden alle Anfragen an OpenStack im Kontext des Benutzers ausgeführt, der gerade eingeloggt ist. Dies stellt für die normale Benutzung kein Problem dar. Für KaaS ist dies allerdings ein Problem, da die

4 Aufbau und Implementierung

KaaS Buildserver Images nach Glance hochladen müssen. Dies ist aber nur mit validen Benutzerdaten möglich. Aus diesem Grund ist es notwendig, dass für KaaS ein eigener Benutzeraccount angelegt wird. Zusätzlich muss ein Projekt angelegt werden und dem KaaS Benutzer Administratorrechte gegeben werden.

Die Benutzerdaten (Benutzername und Passwort) und die Projektdaten (Projektname und ID) müssen dann von Hand zusammen mit der URL des Keystone-Servers in der Einstellungsdatei eingetragen werden. Diese Benutzerdaten werden dann von KaaS verwendet um die Buildserver zu starten und um den Buildserver die Möglichkeit zum Hochladen von Images nach Glance zu geben. Die KaaS Buildserver können leider nicht direkt in das Projekt des Benutzers der die Kernelbauanfrage gesendet hat hochladen, da die Buildserver dazu die Benutzerdaten des Benutzers wissen müssten. Daher wird ein Umweg über das KaaS Projekt gemacht: Das Image wird in das KaaS Projekt hochgeladen und dann über Glance mit dem eigentlichen Benutzer geteilt. Dieser hat dann die Möglichkeit das Image zum Starten von VMs zu benutzen, kann aber leider das Image nicht löschen oder umbenennen. Deshalb wird über die Imageverwaltung im Übersichtsbereich diese Funktionalität nachgebaut indem intern die Benutzerdaten des KaaS Benutzers zum Umbenennen und Löschen der Images verwendet werden.

4.6.1 KaaS intern

Wie die drei KaaS Teile miteinander kommunizieren soll im Folgenden am Beispiel eines Kernelbaus gezeigt werden. Der Benutzer loggt sich in OpenStack ein und startet eine VM basierend auf der KaaS Nutzer VM. Er installiert seine benötigte Software, startet seinen angebotenen Dienst und beginnt dann damit diesen Dienst unter Last zu setzen. Die gestartete VM taucht auf seiner Übersichtsseite auf und bietet die Möglichkeit zum Kernelbau an. Innerhalb der VM läuft seit dem Start der Tracer und dokumentiert alle aufgerufenen Kernelfunktionen.

Der Benutzer kann nun zu jedem Zeitpunkt versuchen einen Kernelbau zu starten. Dazu klickt er in seiner Übersichtsseite auf den *Build a Kernel* Knopf. KaaS schaut nun nach, ob die zuletzt getracete Funktion mehr als einen Tag alt ist. Ist dies der Fall, kann der Benutzer eine Kernelbauanfrage absenden. Ist die zuletzt getracete Funktion innerhalb der letzten 24 Stunden gesehen worden, warnt KaaS vor dem Erstellen eines Kernels. Der Benutzer kann aber trotzdem eine Kernelbauanfrage absenden, wenn er die Warnung durch bestätigen einer Checkbox zur Kenntnis nimmt. Weiterhin hat der Benutzer die Möglichkeit den Tracer in seiner VM weitertracen zu lassen.

Der Ablauf des Kernelbaus und die Kommunikation der einzelnen KaaS Komponenten mit OpenStack wird in Abbildung 4.9 gezeigt. Die Kernelbauanfrage landet zuerst im KaaS Horizon Modul. Dort wird als erster Schritt der Tracer in der Nutzer VM gestoppt, es sei denn es wurde vom Benutzer gewünscht, dass weitergetraced wird. Als zweites sucht das Horizon Modul einen Buildserver für den Kernelbau. Dazu werden alle Buildserver nach ihrem aktuellen Status gefragt.

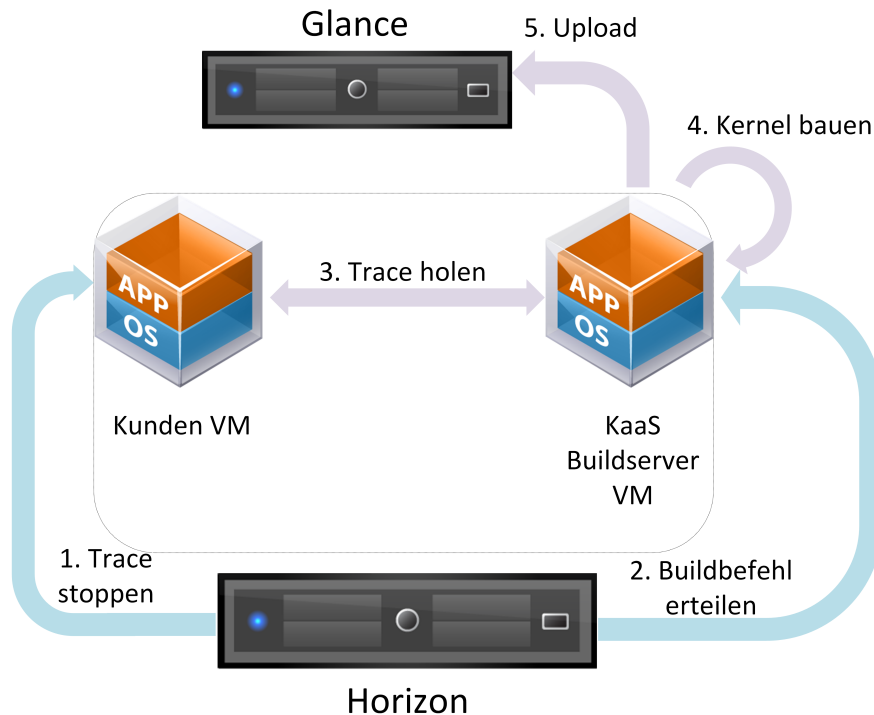


Abbildung 4.9: Interner Ablauf eines Kernelsbaus in KaaS

Berichtet ein Buildserver, dass er zurzeit keinen Kernelbau durchführt, wird dieser Buildserver ausgewählt und kein weiterer befragt. Sind alle Buildserver mit einem Kernelbau beschäftigt, wird der Buildserver mit der kürzesten Warteschlange ausgewählt. Dem ausgewählten Buildserver werden die ID und IP-Adresse der VM und der Projektname des Benutzers mitgeteilt und der Kernelbauauftrag wird in die Warteschlange eingereiht.

Ist der Kernelbauauftrag am Anfang der Warteschlange angekommen, wird die Protokolldatei von der Nutzer VM abgeholt und auf dem Buildserver abgespeichert. Nun wird der Kernelbau gestartet, indem erst von undertaker die Kernelkonfiguration erstellt wird. Danach wird der neue Kernel mit der erstellten Konfiguration gebaut und im Standard Image installiert. Als letztes wird dieses Image nach Glance ins KaaS-eigene Projekt hochgeladen und dann mit dem Projekt des Benutzers geteilt. Der Kernelbau ist damit beendet.

5 Probleme während der Entwicklung

Während der Entwicklung von KaaS sind einige Probleme mit dem Buildserver und den erstellten Images aufgetreten deren Lösung im Folgenden erläutert wird.

5.1 Undertaker konnte keine valide Kernelkonfiguration erstellen

Der erste große Fehler trat während der Entwicklung des Buildserver Images auf. Das Übertragen der Protokolldatei von der KaaS Nutzer VM funktionierte fehlerfrei, allerdings gelang es undertaker nicht eine valide Kernelkonfiguration aus der Protokolldatei zu erstellen. Die einzige Fehlermeldung war ein simples „Wasn't able to generate a valid configuration“. Die Fehlersuche war dementsprechend mühselig obwohl die Lösung sehr einfach ist.

Die KaaS Nutzer VM basiert auf einem Ubuntu 12.04 Daily Build mit Kernelversion 3.2.0-45-virtual. Der Buildserver brauchte die passenden Debugbinaries und Quellcodedateien. Beides wird über `apt-get` bezogen. Allerdings bietet Ubuntu die Kernelsources nur für die aktuellste Version an. Dies bedeutet, dass wenn Version 3.2.0-45-virtual angefragt wurde, aber 3.2.0-52 die aktuelle Version ist, man diese auch bekommen hat. Dies wird von `apt-get` auch so mitgeteilt, geht aber zwischen den anderen Ausgaben leicht unter. Undertaker hat also versucht die Funktionsnamen an den passenden Stellen im Quellcode zu finden und konnte dies nicht erfolgreich tun. Die Debugbinaries waren zwar die richtige Version, aber diese passten nicht zu den Quellcodedateien.

Die korrekte Version des Kernelquellcodes erhält man nur über das git Repository von Ubuntu. Hier lassen sich dann alle Versionen auschecken die benötigt werden.

5.2 Kernelinstallation in einer chroot Umgebung

Das zweite große Problem trat ebenfalls während der Entwicklung des Buildservers auf. Das Ubuntu Image in welchem der neue Kernel installiert werden soll, liegt als Imagedatei vor. Diese Imagedatei ist letztendlich nur ein Festplattenabbild inklusive Partitionen. Die erste Partition innerhalb der Datei wird dann

5 Probleme während der Entwicklung

eingebunden um Zugriff auf das Dateisystem zu erhalten. Der neue Kernel liegt als `.deb` Datei vor und lässt sich so über `dpkg` installieren. Um den Buildserver nicht zu verändern wird eine `chroot` Umgebung verwendet, da so nur auf dem eingebundenen Dateisystem gearbeitet wird. Allerdings nimmt `dpkg` an, dass man versucht den Kernel für das aktuell laufende System zu installieren und versucht deshalb die GRUB Konfiguration zu aktualisieren. Dabei scheitert das Installationsskript daran herauszufinden auf welcher Festplatte das System installiert ist, da es sich ja in einem `chroot` befindet.

Der erste Lösungsansatz war das Installationsskript anzupassen und die GRUB Konfiguration selbst zu bearbeiten. Leider stellt sich beim Entpacken des `.deb` Archives heraus, dass gar kein Installationsskript enthalten war. Dieses muss also von einer anderen Quelle kommen. Der zweite Lösungsansatz war dann der Versuch dem Installationsskript die passenden Informationen so mitzuteilen, dass es in der Lage ist eine gültige GRUB Konfiguration zu erstellen. Dies erwies sich als nicht trivial, da das Skript ja nicht bekannt ist. Es wurde versucht dem Skript durch extra erstellte `/etc/mtab` und `/boot/grub/device.map` Dateien die richtige Festplatte mitzuteilen. Dies schien zu funktionieren da jetzt das Installationsskript ohne Fehler abschloss. Leider scheiterten alle Versuche dann am booten des neuen Images. Die erstellte GRUB Konfiguration war leider nicht korrekt.

Die endgültige Lösung besteht nun aus dem Entpacken des `.deb` Archives und Anpassung der GRUB Konfiguration von Hand. Dazu werden einfach nur die Versionsnummern der Kernel in der GRUB Konfiguration so ausgetauscht, dass nicht mehr der alte, sondern der neue Kernel gestartet wird. Weiterhin wird `update-initramfs` aufgerufen, um eine neue Ramdisk zu erstellen. Dieses Schritt wurde ursprünglich vom Installationsskript übernommen. Abschließend werden die Symlinks zum Kernel und zur Ramdisk auf die neuen Versionen umgestellt.

5.3 Aufhängen beim Boot durch OpenStack

Nach erfolgreicher Kernelinstallation und anpassen der GRUB Konfiguration trat das nächste Problem auf. Sobald die Images in OpenStack gestartet wurden, hängte sich die VM direkt nach dem GRUB Menü auf und konsumierte alle verfügbare CPU-Zeit. Mit den Recovery Optionen bootete die VM allerdings einwandfrei in den neuen Kernel, was zeigt, dass dieser korrekt installiert wurde.

Das Ursache war recht simpel: OpenStack bietet die Möglichkeit von Horizon aus Zugriff auf den Log der VM zu erhalten. Dazu wird über eine Bootoption die Ausgabe der VM von der Konsole auf die serielle Schnittstelle weitergeleitet. OpenStack fragt dann diese Schnittstelle ab und bezieht so den Log der VM. Im Recovery Modus wird dies nicht gemacht. Nach entfernen der Option für den Normalboot bootete die VM auch wieder normal.

Die Lösung ist allerdings nicht das entfernen der Option, sondern das Hinzufügen einer weiteren. Die GRUB Konfiguration wird vom Buildserver nun so

5.3 Aufhängen beim Boot durch OpenStack

angepasst, das auf die VM auf die normale Konsole und auf die serielle Schnittstelle logt. Damit bootet die VM in jedem Fall, selbst wenn es Probleme mit der seriellen Schnittstelle gibt. Warum die VM sich nicht starten lässt, wenn nur auf die serielle Schnittstelle geloggt wurde, konnte leider nicht geklärt werden.

6 Evaluation

Es ist anzunehmen, dass Kernel maßschneidern keine Auswirkungen auf die Leistung des Gesamtsystems hat, da durch den Prozess des Maßschneiderns nur unbenutzte Features entfernt werden. Um die These zu überprüfen wurde eine Evaluation am Beispiel eines `memcached` Servers durchgeführt. Hierzu wurde eine OpenStack Installation (in Form von DevStack¹) auf einem Ubuntu 13.04 durchgeführt und um das Kernel as a Service Horizon Modul erweitert. Der physische Rechner ist ein Dell Optiplex 7010 mit einem Intel i7-3770 Prozessor, 16GB Ram und einer 256GB SSD.

Um die Performance von `memcached` zu messen, wird auf das Programm `memslap` zurückgegriffen, welches mit `libmemcached-tools` mitgeliefert wird. `memslap` führt einen Test mit einer fixen Anzahl an Anfragen durch und gibt die Zeit zurück, die vom `memcached` Server benötigt wird, um alle Anfragen zu beantworten. Um äußere Einflüsse so gering wie möglich zu halten, läuft jede zu testene VM alleine, das heißt es gibt keine anderen VMs die Ressourcen beanspruchen könnten. Für die Messungen werden drei verschiedene VMs gemessen:

1. Ein unverändertes Ubuntu 12.04. Kein installierter und laufender Tracer. Basiert auf dem Ubuntu 12.04 Daily Build Cloud Images
2. Die KaaS Nutzer VM mit installiertem und gestartetem Tracer.
3. Ein Ubuntu 12.04 mit maßgeschneidertem Kernel. Der Kernel wurde aus den Tracedaten der erwähnten KaaS Nutzer VM erstellt.

Alle VMs wurden mit dem OpenStack Flavor `m1.small` erstellt. Dies entspricht einer vCPU und 2GB Arbeitsspeicher. Der `memcached` Server wurde auf allen VMs mit folgendem Befehl gestartet: `memcached -m 1024`. Dies reserviert die Hälfte des verfügbaren Arbeitsspeichers für `memcached`. Verwendet wurde Version 1.4.13 von `memcached`. `memslap` wurde nicht in der VM gestartet, sondern auf dem physischen Host und kommuniziert mit der `memcached` VM über das virtuelle Netzwerk. Weiterhin wurden pro Versuch 10 Durchläufe gemacht um einen Durchschnittswert zu ermitteln. Mit jeder `memcached` VM wurden 6 Versuche mit unterschiedlicher Anzahl an simulierten Nutzern durchgeführt. `memslap` wurde dazu mit folgenden Argumenten aufgerufen: `memslap -flush -binary -concurrency X` Für X wurde die jeweils die Anzahl der zu simulierenden Nutzer eingesetzt.

¹Siehe auch <http://devstack.org/>

6.1 Ergebnisse der Performanzanalyse

Als erste VM kam die unveränderte Ubuntu VM zum Einsatz. In der folgenden Tabelle 6.1 sind die Ergebnisse der Messungen zu finden.

Tabelle 6.1: Tastdauer des `memcs1ap` Benchmarks für ein unverändertes Ubuntu 12.04. Angabe in Sekunden; weniger ist besser

# Nutzer	min	max	avg	σ
1	0,973	1,096	1,027	0,033
5	0,794	1,039	0,901	0,061
10	1,257	1,662	1,421	0,135
25	2,858	3,937	3,245	0,323
50	5,013	5,855	5,456	0,320
100	8,915	10,423	9,616	0,495

Als zweite VM wurde die KaaS Nutzer VM gemessen. Diese unterscheidet sich nur durch den installierten Kerneltracer und den KaaS Python Trace Server von dem unveränderten Ubuntu. Die Messergebnisse folgen in Tabelle 6.2.

Tabelle 6.2: Tastdauer des `memcs1ap` Bechmarks für eine KaaS Nutzer VM (basierend auf Ubuntu 12.04). Angabe in Sekunden; weniger ist besser

# Nutzer	min	max	avg	σ
1	0,739	1,066	0,970	0,094
5	0,806	1,020	0,871	0,066
10	1,186	1,472	1,356	0,077
25	2,595	3,769	3,147	0,359
50	4,730	5,968	5,288	0,353
100	8,722	9,796	9,225	0,310

Wie aus beiden Tabellen zu entnehmen ist, erzeugt das Tracen keinen messbaren Overhead und verringert die Performanz des `memcached` Servers nicht. In Tabelle 6.3 folgt nun das Testergebniss der VM mit maßgeschneidertem Kernel.

Im Vergleich mit den Testergebnissen der beiden anderen VMs zeigt sich auch hier kein Performanzunterschied des `memcached` Servers. Alle Ergebnisse liegen innerhalb der natürlichen Schwankungen. Abbildung 6.1 zeigt noch einmal die Ergebnisse im direkten Vergleich.

Es lässt sich festhalten, dass durch den im Hintergrund laufenden Tracer kein messbarer Leistungsverlust zu verbuchen ist. Weiterhin führt auch ein maßgeschneiderter Kernel zu keinem messbaren Leistungsunterschied. Die kleinen Unterschiede im Mittelwert der Ergebnisse lassen sich auf die Schwankungen in der Messung zurückführen, wie an der Standardabweichung zu sehen ist.

6.2 Vergleich eines maßgeschneiderten und generischen Kernels

Tabelle 6.3: Tastdauer des `memcs1ap` Bechmarks für eine VM mit maßgeschneidertem Kernel in einem Ubuntu 12.04. Angabe in Sekunden; weniger ist besser

# Nutzer	min	max	avg	σ
1	0,454	0,973	0,809	0,199
5	0,759	0,988	0,902	0,074
10	1,166	1,369	1,297	0,062
25	2,761	4,138	3,344	0,405
50	4,939	5,903	5,375	0,274
100	8,545	11,041	9,225	0,735

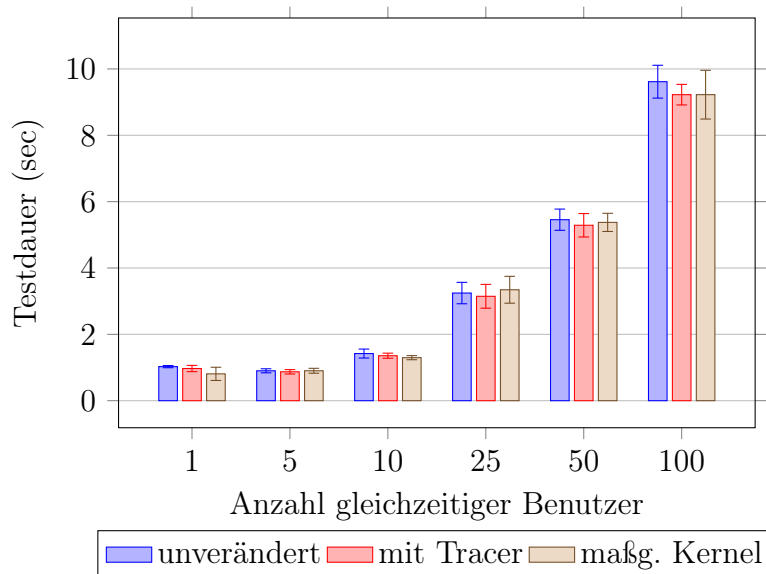


Abbildung 6.1: Testergebnisse aller VMs im direkten Vergleich; weniger ist besser

6.2 Vergleich eines maßgeschneiderten und generischen Kernels

Wie bereits gezeigt, unterscheiden sich ein maßgeschneiderter Kernel und ein generischer Kernel nicht in ihrer Leistung. Beide Kernel unterscheiden sich aber in ihren bereitgestellten Features. Im undertaker Paper[1] wurden Analysen für einen NFS Server und einen LAMP Server durchgeführt. Für den maßgeschneiderten Kernel des LAMP Servers ergab sich dabei eine drastische Verkleinerung des Kernels, wie in Abbildung 6.2 zu sehen ist. Entfernt wurden sehr viele unbenutzte Gerätetreiber, aber auch ungenutzte Dateisystemunterstützung und Kryptografiefunktionen wurden entfernt. Das im generischen Kernel vorhandene Soundsystem wurde komplett ausgebaut.

6 Evaluation

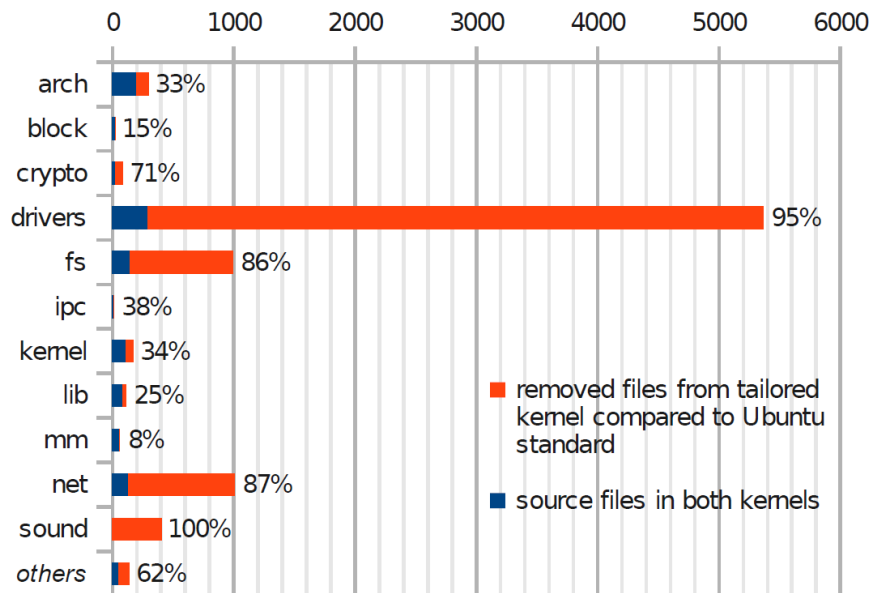


Abbildung 6.2: Verringerung der kompilierten Quellcodedateien für den maßgeschneiderten Kernel im Vergleich zum Ubuntu Standard Kernel für einen LAMP Server
Quelle: [1]

Für den im vorherigen Abschnitt evaluierten `memcached` Server wurde auch ein Vergleich beider Kernel gemacht. Dazu wurde die `virtual` Version als generischer Kernel genommen (wie in den Daily Cloud Images) und dann maßgeschneidert. Kompiliert wurde der generische Kernel mit der Kernelkonfiguration, die den Ubuntu Cloud Daily Images beilag. Das Ergebnis ist in [Abbildung 6.3](#) und [Tabelle 6.4](#) zu sehen.

Hier zeigt sich, wie beim `undertaker` Beispiel, ein sehr ähnliches Bild. Hauptsächlich Gerätetreiber und Dateisystemunterstützung wurde entfernt. Wie zu erkennen ist, wurde auch das Soundsystem wieder ausgebaut. Dies ist ein interessanter Punkt, da der Ausgangskernel bereits das Suffix `virtual` trägt. Dies signalisiert, dass der Kernel bereits für virtuelle Server angepasst worden ist, indem unnötige Treiber und Subsysteme entfernt worden sind[11]. Warum das Soundsystem (bestehend aus 626 Dateien) trotzdem mit eingebaut wurde ist verwunderlich. Es ist höchst unwahrscheinlich, dass ein virtueller Server dieses benötigt.

6.2 Vergleich eines maßgeschneiderten und generischen Kernels

Tabelle 6.4: Anzahl der kompilierten Quellcodedateien für verschiedene Bereiche des Linuxkernels

Bereich	original (davon als Modul)	maßg. (davon als Modul)	Abnahme
arch	309 (50)	216 (15)	93 (30,10%)
block	26 (0)	24 (1)	2 (7,69%)
crypto	143 (58)	25 (0)	118 (82,52%)
drivers	7899 (4723)	160 (12)	7739 (97,97%)
fs	1110 (798)	146 (35)	964 (86,85%)
ipc	13 (0)	9 (0)	4 (30,77%)
kernel	171 (0)	118 (0)	53 (30,99%)
lib	141 (34)	85 (1)	56 (39,72%)
mm	60 (0)	56 (0)	4 (6,67%)
net	1356 (844)	135 (14)	1221 (90,04%)
sound	626 (404)	0 (0)	626 (100,00%)
andere	138 (40)	54 (0)	84 (60,87%)

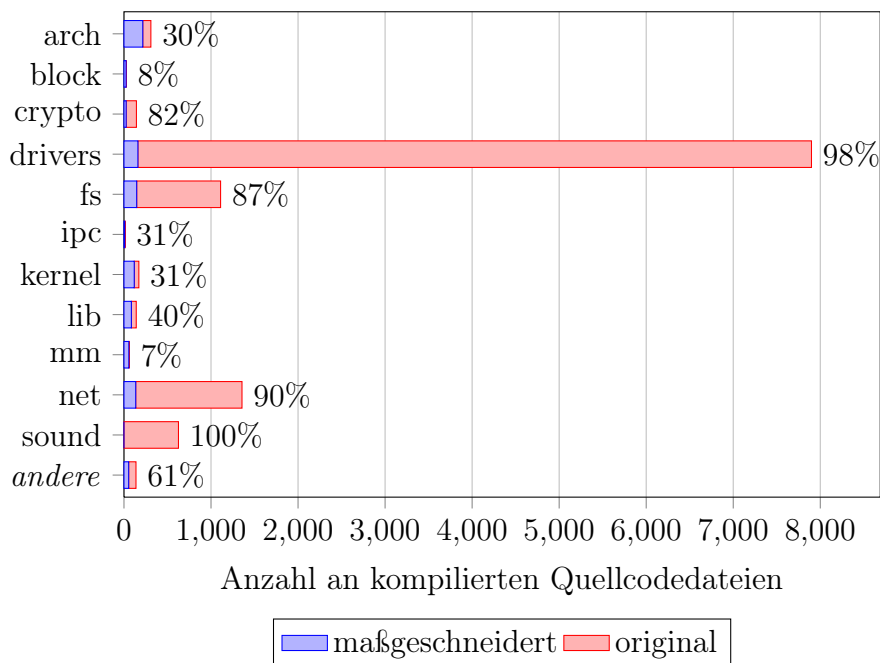


Abbildung 6.3: Verringerung der kompilierten Quellcodedateien für den maßgeschneiderten Kernel im Vergleich zum Ubuntu Standard Kernel für einen memcached Server

6 Evaluation

Um herauszufinden, warum der `virtual` Kernel so viel enthält, wurden die Kernelkonfigurationen der `generic`² Variante mit der Konfiguration der `virtual` Variante verglichen. Es hat sich gezeigt, dass die Optionen für den verwendeten Scheduler und das Multitasking geändert worden sind. Weiterhin wurde die maximale Anzahl an unterstützten CPU Kernen verringert. Außerdem wurden einige Netzwerktreiber nicht mehr direkt in den Kernel sondern als Modul kompiliert. Der Vergleich der beiden Kernelkonfiguration und die wenigen Unterschiede erklären die große Menge an entfernten Features. Warum allerdings nur so wenige Änderungen vollzogen wurden ist unklar, da die `virtual` Variante angeblich von unnötigen Treibern befreit wurde[11].

Abschließend lässt sich sagen, dass ein Maßschneidern von Kernen eine effiziente Methode ist um die Sicherheit eines Systems durch Entfernen von unbenutzten Kernelfeatures zu erhöhen. Dabei sind keine Leistungseinbußen im Vergleich zu einem generischen Kernel zu verzeichnen.

²Verfügbar unter <http://kernel.ubuntu.com/~kernel-ppa/configs/precise/>

7 Ähnliche Ansätze und mögliche Erweiterungen

Die Idee hinter KaaS ist recht simpel: Benutzern von IaaS Clouds die Möglichkeit geben automatisiert maßgeschneiderte Kernel für ihre VMs zu erstellen. Lässt man den Automatisierungsaspekt weg, wird von Amazon EC2 und OpenStack bereits eine ähnliche Lösung angeboten.

Amazon unterscheidet drei verschiedene Image Typen in EC2: *Amazon Machine Image*, *Amazon Kernel Image* und *Amazon Ramdisk Image*. Das Amazon Machine Image enthält dabei das eigentliche Betriebssystem und Anwendungen allerdings ohne den Kernel. Das Machine Image ist also nicht alleine bootfähig. Der Kernel wird im Amazon Kernel Image gespeichert. Dieses ist theoretisch bootfähig, enthält aber kein Betriebssystem welches gebootet werden könnte. Kombiniert man beide Images, entsteht ein bootbares Image, welches in ein Betriebssystem bootet. Im dritten Image, dem Amazon Ramdisk Image, wird die eventuell optionale Ramdisk des Systems abgelegt. Dieses kann man den anderen beiden Images kombiniert werden. Das Verbinden der Images geschieht aber erst beim Erstellen einer virtuellen Maschine. Dies erlaubt es ein Betriebssystem aus einem Machine Image mit unterschiedlichen Kernen zu starten. Die Images werden von Amazon bereitgestellt, allerdings können Kunden auch eigene Images hochladen, benutzen und sogar mit anderen Kunden teilen.

Diese Image Trennung würde ein KaaS ähnliches System sehr einfach machen. Die KaaS Buildserver würden statt ganzen Images nur Kernel Images erstellen und diese den Benutzern zur Verfügung stellen. Der Benutzer kann dann beim Booten einer neuen VM entscheiden, ob er den generischen Kernel oder den maßgeschneiderten Kernel verwenden möchte. Da Benutzer allerdings in der Lage sind ihre eigenen Images hochzuladen, wäre Maßschneidern auch manuell möglich. Kunden sind in der Lage sich ihr eigenes Betriebssystem Image mit installiertem Undertaker zu erstellen und können aus der Protokolldatei dann manuell einen maßgeschneiderten Kernel bauen. Dieser Kernel kann dann als Kernel Image wieder hochgeladen und zum Erstellen von VMs benutzt werden.

OpenStack unterstützt die von Amazon benutzten Imageformate, allerdings mit einem erheblichen Nachteil. Bei Amazon werden Machine und Kernel Image beim Erstellen einer VM verknüpft. Diese Verknüpfung ist allerdings nur für diese VM gültig. Dies bedeutet, dass eine weitere VM mit gleichem Machine- aber anderem Kernel-Image gestartet werden kann. Bei OpenStack findet diese Verknüpfung zwischen Kernel und Machine Image unabhängig von den VMs statt.

7 Ähnliche Ansätze und mögliche Erweiterungen

Dies ist ein erheblicher Nachteil, da Machine Images immer nur mit einem Kernel Image gleichzeitig verknüpft sein können. Möchte man eine VM mit anderem Kernel Image starten, muss zuerst die bestehende Verknüpfung aufgelöst und eine neue erstellt werden. Ist das Machine Image nun mit mehreren Benutzern geteilt, wird die Verknüpfung auch für alle anderen Benutzer geändert.

Dieser Nachteil ist auch der Grund dafür, warum KaaS ganze Images erstellt und dem Benutzer zur Verfügung stellt, anstatt nur Kernel zu bauen und diese als Kernel Images anbietet. KaaS würde in diesem Fall ein Basis Ubuntu Image anbieten und die maßgeschneiderten Kernel Images mit diesem verknüpfen. Leider wirkt sich dies dann auch auf alle anderen Benutzer aus indem bereits gemachte Verknüpfungen aufgelöst werden. Bei Amazon EC2 ist dieses Problem nicht vorhanden, da die Verknüpfung zur VM gehört und nicht zum Machine Image.

7.1 Kernel verkleinern mit Linux Boardmitteln

Der Linux Kernel lässt sich mit über 11000 Konfigurationsoptionen sehr stark anpassen. Die richtigen Optionen auszuwählen ist keine einfache Aufgabe. Das bereits vorgestellte Programmpaket `undertaker` hilft dabei, indem nur benötigte Features aktiviert werden. Dazu benutzt `undertaker` `ftrace`, ein Kernelprogramm um Funktionsaufrufe im Kernel zu protokollieren. `Undertaker` kann mit Hilfe der protokollierten Funktionen eine Kernelkonfigurationsdatei erstellen, die nur noch Features aktiviert, dessen Funktionen aufgerufen wurden.

Allerdings ist es auch mit Linux Boardmitteln möglich einen maßgeschneiderten Kernel zu erstellen. Lädt man sich den Linux Kernel Quellcode herunter, liegt diesem eine Makefile bei. Die Makefile dient zum vereinfachten Bauen des Linux Kernels. Andererseits ist es auch möglich die Kernelkonfiguration mit Hilfe der Makefile zu verändern. Über den Befehl `make allmodconfig` ist es möglich sich eine Kernelkonfiguration zu erstellen, die alle möglichen Features des Kernels als Kernelmodul erstellt. Beim Booten des Kernel werden dann nur die Module geladen, die auch tatsächlich benötigt werden. Weiterhin ist es auch möglich, dass weitere Module während der Laufzeit nachgeladen werden.

Installiert nun ein Benutzer seine Anwendung und setzt sie unter Last, gibt es wie beim Kerneltracing einen Zeitpunkt, ab dem keine neuen Module mehr nachgeladen werden. Nun lässt sich mit `make localmodconfig` eine neue Kernelkonfiguration erstellen, die nur noch die aktuell geladenen Module aktiviert und alle anderen Module ausbaut. Durch dieses Verfahren lassen sich einfach maßgeschneiderte Kernel erstellen, ohne dabei auf externe Programme angewiesen zu sein. Aus Zeitgründen konnte leider kein Vergleich zwischen einem durch KaaS maßgeschneiderten und einem mit `make localmodconfig` maßgeschneiderten Kernel durchgeführt werden.

7.2 Mögliche Erweiterungen

Die Entwicklung von KaaS ist zwar abgeschlossen, doch es kamen bereits diverse Ideen für mögliche Erweiterungen des KaaS Systems auf. Zwei dieser Ideen werden im Folgenden erläutert.

Aktuell erzeugt KaaS für jede ankommende Kernelbauanfrage einen neuen Kernel, installiert diesen in einem Image und lädt dieses hoch. Bei einer kleinen Nutzerbasis und unterschiedlichen VMs kann dies noch gut funktionieren. Sollte KaaS aber mit einer großen Nutzerbasis (Größenordnung Rackspace/Amazon EC2) benutzt werden, kommt es unweigerlich vor, dass Benutzer VMs für exakt gleiche Zwecke benötigen. Für jede dieser VMs wird ein eigener Kernel gebaut, obwohl ein bereits bestehender Kernel genutzt werden könnte. Somit lässt sich Zeit und auch Platz sparen, da ein bereits fertiges Image nun nicht mehr nur einem, sondern mehreren Parteien zugänglich gemacht wird.

Problematisch ist allerdings herauszufinden wann ein bestehender Kernel ausreicht. Eine Liste der installierten Software ließe sich leicht über `dpkg` abfragen. Es lässt sich annehmen, dass VMs mit gleicher installierter Software auch den gleichen Kernel benötigen. Dies ist leider nicht ganz so einfach. Beispielsweise gibts es zwei VMs mit einem LAMP-Stack¹. Beide Server benutzen PHP-Skripte um diverse Aufgaben zu erledigen. Die Skripte des einen Server benutzen intern allerdings Systemfunktionen die bestimmte Kernelfeatures benötigen. Die PHP-Skripte des anderen Servers nicht. Ein simpler Vergleich der `dpkg` Software Liste zeigt aber gleiche Software auf beiden Servern. Der maßgeschneiderte Kernel für den zweiten Server kann aber nicht für den ersten Server verwendet werden, da benötigte Kernelfeatures fehlen.

Eine mögliche Lösung basiert auf dem Vergleich der von undertaker erstellten Kernelkonfigurationsdateien. Ein Kernel erstellt mit einer Konfigurationsdatei kann auch verwendet werden, wenn ein anderer Trace zu einer gleichen Konfiguration führt. Beide erstellten Kernel würden mit den gleichen Features gebaut werden und damit identisch sein. KaaS wäre so zu erweitern, dass zu jedem erstellten Kernel die Konfigurationsdatei mit gespeichert wird. Bei jeder Kernelbauanfrage würde dann aus der Protokolldatei eine Kernelkonfiguration erzeugt werden und diese dann mit allen anderen bisher vorhandenen Konfigurationen verglichen werden. Dies würde eine zentrale Datenbank erfordern, die die Zuordnungen von Kernelkonfigurationen und maßgeschneiderten Kernen abspeichert. Weiterhin müssten alle KaaS Buildserver Zugriff auf diese Datenbank haben.

Wie beschrieben wäre es prinzipiell möglich ein Wiederverwenden der erstellten Images zu realisieren. Allerdings ist diese Funktionalität sehr komplex und wurde daher nicht mit eingebaut.

¹Linux, Apache, MySQL, PHP; also ein Webserver mit Datenbank

7.2.1 CVE Alarme

Eine weitere Ergänzung zu KaaS wäre die Möglichkeit Benutzer mit Standardkernen vor *Common Vulnerabilities and Exposures* (CVE) zu warnen von denen sie betroffen sind. Das CVE System weist Sicherheitslücken eine eindeutige Zahlenfolge zu unter der dann eine Beschreibung der Lücke zu finden ist. Sicherheitslücken des Linuxkernels können so eindeutig identifiziert werden. Die Erweiterung von KaaS basiert nun auf den frei verfügbaren CVE Listen. Benutzer die einen generischen Kernel einsetzen können mithilfe der CVE vor Sicherheitslücken in ihrem Kernel gewarnt werden, die durch Maßschneidern der Kernels eventuell entfallen.

Eine weitere mögliche Erweiterung würde auch Benutzern von bereits maßgeschneiderten Kernen behilflich sein. Tauchen CVEs auf, die Bereiche betreffen welche im maßgeschneiderten Kernel enthalten sind, kann der Benutzer dazu aufgefordert werden auf eine neue Kernelversion upzudaten. Hierzu müssten allerdings die KaaS Buildserver so ausgebaut werden, dass sie mehr als eine Kernelversion kompilieren können. Ein weiteres Problem sind die CVEs selbst. Die angebotene Beschreibung sagt nicht immer eindeutig aus in welchem Bereich des Kernel die Sicherheitslücke liegt, bzw. durch welche Konfigurationsoptionen der Bereich aktiviert wird. Weiterhin werden CVEs nur selten mit einem Fix für die Sicherheitslücke ausgegeben, wodurch erstmal eine Wartezeit eingeplant werden muss, bevor ein Fix entwickelt und im Kernel eingebaut wurde.

8 Zusammenfassung

Infrastructure as a Service Anbieter bieten ihren Kunden Templates an, die mit einem generischen und feature-reichen Linux Kernel ausgestattet sind, um möglichst hohe Kompatibilität zu erreichen. Allerdings bieten solche umfangreichen Kernel möglicherweise viele Sicherheitslücken.

Es bildet sich jedoch ein Trend zu spezialisierten virtuellen Maschinen, wodurch nur ein kleiner Teil der bereitgestellten Kernelfeatures benötigt wird. Doch selbst in nicht benötigten Kernelfeatures können Sicherheitslücken ausgenutzt werden.

Das im Rahmen dieser Bachelorarbeit entwickelte *Kernel as a Service* (KaaS) wirkt dem entgegen, indem es die durch Anpassen der Konfigurationsoptionen des Linuxkernels diesen auf die Bedürfnisse der Anwendung maßschneidert. KaaS ist dabei besonders benutzerfreundlich, weil es virtuelle Maschinen selbstständig analysiert und die Konfiguration eines Kernels automatisiert.

Die Architektur von KaaS gliedert sich in drei Teile: Das KaaS Horizon Modul stellt die Verwaltungsoberfläche bereit und integriert sich in OpenStacks Horizon. Weiterhin wird für die Anwender ein speziell angepasstes VM Template bereitgestellt, in welchem das Kerneltracerprogramm *undertaker* arbeitet und Funktionsaufrufe im Kernel protokolliert. Eine weitere, von KaaS bereitgestellte, VM übernimmt das Maßschneidern der Kernel.

Benutzer von KaaS starten das bereitgestellte VM Image mit installiertem Kerneltracer. Im Hintergrund kümmert sich *undertaker* dann um das Aufzeichnen aller aufgerufenen Kernelfunktionen. Werden keine neuen Kernelfunktionen mehr gefunden, wird vom Benutzer ein Kernelbau in Auftrag gegeben. Der gesamte Prozess des Kernelbauens läuft automatisch und ohne Interaktion mit dem Benutzer ab. Die fertigen Images mit neuem Kernel werden wieder über OpenStacks Image Service Glance verfügbar gemacht.

Die Evaluation des Systems wurde am Beispiel eines `memcached` Servers durchgeführt. Es konnte gezeigt werden, dass durch den Einsatz eines maßgeschneiderten Kernels ein großer Teil der Features entfernt werden konnte. Somit konnte im Vergleich zum generischen Kernel die Angriffsfläche stark verringert werden. Außerdem konnte gezeigt werden, dass das Maßschneidern keine Auswirkungen auf die Performanz des Kernels hat.

Literaturverzeichnis

- [1] Anil Kurmus, Reinhard Tartler, Daniela Dorneanu, Bernhard Heinloth, Valentin Rothberg, Andreas Ruprecht, Wolfgang Schröder-Preikschat, Daniel Lohmann, and Rüdiger Kapitza. Attack Surface Metrics and Automated Compile-Time OS Kernel Tailoring. In *Proceedings of the 20th Network and Distributed System Security Symposium*, 2013.
- [2] OpenStack, Cloud Software. <http://www.openstack.org/>. Abgerufen am: 21.10.2013.
- [3] Undertaker Trac. <http://vamos.informatik.uni-erlangen.de/trac/undertaker>. Abgerufen am: 24.09.2013.
- [4] Eucalyptus, Open Source AWS Compatible Private Clouds. <http://www.eucalyptus.com/>. Abgerufen am: 21.10.2013.
- [5] OpenNebula, Flexible Enterprise Cloud Made Simple. <http://opennebula.org/>. Abgerufen am: 21.10.2013.
- [6] Companies Supporting The OpenStack Foundation. <http://www.openstack.org/foundation/companies/>. Abgerufen am: 17.09.2013.
- [7] Nova API Feature Comparison. <https://wiki.openstack.org/wiki/Nova/APIFeatureComparison>. Abgerufen am: 17.09.2013.
- [8] Swift API Feature Comparison. <https://wiki.openstack.org/wiki/Swift/APIFeatureComparison>. Abgerufen am: 17.09.2013.
- [9] OpenStack Object Storage (Swift). http://en.wikipedia.org/wiki/OpenStack#Object_Storage_.28Swift.29, 2013. Abgerufen am: 21.10.2013.
- [10] Service Name and Transport Protocol Port Number Registry. <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=16>, 2013. Abgerufen am: 03.10.2013.
- [11] Frequently Asked Questions about the Ubuntu Server Edition. https://help.ubuntu.com/community/ServerFaq#What_are_the_differences_between_the_server_and_virtual_kernels.3F. Abgerufen am: 23.10.2013.