Nimbus Infrastructure

Daniel Defiebre

Hochschule Mannheim
Fakultät für Informatik
Paul-Wittsack-Straße 10
68163 Mannheim
daniel.defiebre@freenet.de

Zusammenfassung Nimbus Infrastructure ist eine Private Cloud Lösung bassierend auf Linux. Dabei setzt Nimbus für die Virtuallisierung auf XEN oder KVM. Virtuelle Machinen können dann über einen Client auf die Nimbus Infrastruktur übertragen werden und dort ausgeführt werden. Zusätzlich bietet Nimbus einen Speicherdienst an, auf dem die Virtuellen Maschinen abgelegt werden können.

Virtuallisierungstechniken sind heute der Standart von isolierten Applikationsumgebungen. Eine Cloud basierte Virtuallisierung bietet den Vorteil, VM Images auf verschiedene Rechner zu verteilen, um so eine optimale Auslastung zu gewinnen. Darüber hinaus bietet eine Cloud Infrastrukur auch Verwaltungstools für das Manager der Virtuellen Maschinen. Dabei können die Clients, diese die Images managen, einfach über das HTTP Protokoll mit der Infrastruktur kommmunizieren. Nimbus Infrastructure bietet eine kostenlose Lösung zur Umsetzung einer Cloud Infrastruktur auf vorhandenen Rechnerverbunden.

1 Allgemeines zu diesem Dokument

Dieses Dokument beschreibt die Installation und Handhabung von Nimbus Infrastruktur. Dieses Dokument geht hier verstärkt auf Problematiken der Installation ein diese nicht in der Nimbus Dokumentation angesprochen werden. Ergänzend können die Dokumentationen auf der Nimbus Seite www.nimbusproject.org hinzugefügt werden.

2 Übersicht

Nimbus Infrastructure ist eine private Cloud Lösung. Die Strukturierung der Infrastruktur sieht folgende Module vor:

2.1 Cloud Client

Der Rechner von dem aus auf die Infrastruktur zugegriffen wird. Hier wird das Client Programm installiert und die Zertifikate müssen übertragen werden, damit der Client zum Service Node verbinden kann. Das Protokoll ist hierbei http.

2.2 Service Node

Auf dem Service Node befindet sich der Dienst von Nimbus. Dieser ist in Java und Python geschrieben und benötigt noch zusätzlich andere Abhängigkeiten. Zusätzlich besitzt der Service Node den Cumulus Storage. Dies ist ein Speicher auf dem die Virtuellen Maschinen gespeichert werden können. Eine Virtuelle Maschine wird, wenn sie ausgeführt werden soll, auf einen VMM Node kopiert und dort ausgeführt. Der Service Node kommuniziert mit dem VMM Node mit Zertifikate über SSH, da hier eine automatische Verbindung nötig ist.

2.3 VMM Node

Auf dem VMM Node befindet sich der Virtual Maschine Monitor. Dabei unterstützt Nimbus sowohl XEN als auch KVM. Zusätzlich benötigt der VMM Node das Paket libvirt dieses eine virtualisierungs- API ist um auf den VMM zu zu greifen. Das Nimbus Paket workspace-control muss zusätzlich installiert werden, da dieses Paket die Verbindungen zum Service Node bereitstellt und das Einbinden der Virtuellen Maschinen über libvirt kontrolliert.

2.4 DHCP Server

Damit die VMS IP Adressen und Hostname bekommen ist ein DHCP zuständig. Dabei ist die Übergabe von IP Adressen und Hostnamen voll dynamisch. Nimbus selber hat auf dem Service Node einen vorreservierten Adress- Pool diesen er für die Netzwerke Zuweisung über DHCP verwenden kann.

3 Installation

Dieser Abschnitt beschreibt die einzelnen Installationen

3.1 Service Node Abhängigkeiten

Test Systeme: Debian, Ubuntu, SuSE

Unter Debian und Ubuntu v11 war die Installation des Service Nodes problemlos, nur lediglich SuSE machte aufgrund der Paketabhängigkeiten Probleme. Ubuntu Version 12 brach bei einem unbekannten Fehler am Ende der Installation ab obwohl die Pakete hier vorlagen

Folgende Pakete sind notwendig damit Nimbus Service installiert werden kann: Java jdk und jre ab Version 1.5. In den Testumgebungen wurde v 1.6 installiert. Nimbus empfiehlt Sun Java aber mit open Java gab es keine Probleme. Python 2.5 oder höher. In den Testumgebungen wurde Python 2.7 installiert. Dies ist die aktuelle Version der 2er Generation. Nimbus unterstützt nicht Python 3.

Python Zusatzpaket. Nimbus benötigt zusätzliche Pakete wie Splite 2 und sqlite3. Diese sind nötig für die Kommunikation mit den Datenbanken von Nimbus. Andere Pakete: aufgrund der Kommunikation zwischen Service Node und VMM Node brauch Nimbus Open SSL für passwortfreie Kommunikation über SSL und libssl-dev für die Kommunikation zwischen Python und SSL. Zusätzlich benötigt Nimbus Service auch Apache Ant 1.6 oder höher.

3.2 Service Node Installation des Services

Zuerst wird das Paket des Services von der Nimbus Seite geladen mit:

`$ wget http://www.nimbusproject.org/downloads/nimbus-iaas-2.9-src.tar.gz`
Danach werden die Berechtigungen geändert und entpackt

`$ tar xfz nimbus-iaas-2.9-src.tar.gz` Es wird ein Ordner nimbus-2.9-src/ angelegt, indem sich ein Installations- Script befindet.

Dies wird nun ausgeführt mit

`$ ./install /home/nimbus/NIMBUS_HOME`

Anmerkung: Die Installation wird nicht im Sudo Modus ausgeführt.

Nach Fertigstellung der Installation, wurde ein Nimbus System installiert mit integrierter Security, Web Services und Konfiguration Standards. Dies ist aber lediglich ein eine temporäre Konfiguration um die Installation testen zu können.

Die Default Einstellungen für die Netzwerk Ports der Dienste sind:

```
Cumulus : 8888
Nimbus Service Interface : 8443
EC2 compatible Query Interface: 8444
```

Es wird ein x509 Zertifikat erstellt. Später werden eigene Zertifikate erstellte.

Zunächst werden die Installierten Services getestet:

```
nimbus@ubuntu:~/NIMBUS_HOME$ ./bin/nimbusctl start
Launching Nimbus services...   OK
Launching Cumulus services...  OK
```

Nun wird der erster User mit default Werten angelegt um später auf das System zu zu greifen:

```
nimbus@ubuntu:~/NIMBUS_HOME$ ./bin/nimbus-new-user -d /tmp/nimbususer nimbus@nimbus.com
cert            : /tmp/nimbususer/usercert.pem
key             : /tmp/nimbususer/userkey.pem
dn              : /O=Auto/OU=b4b8062d-5ca2-4d06-b1ec-3d38ca8d53ef/CN=nimbus@nimbus.com
canonical id    : ad89fe98-8401-11e1-bb87-000c29a82c65
access id       : ONBeIGfEIUCggkzfjAOIT
access secret   : 8BVWYWfSQPfHAGtFIj0mD0c3zKmuwZ1PB72gIdKXAJ
url             : None
web id          : None
cloud properties : /tmp/nimbususer/cloud.properties
```

3.3 Nimbus Client Installation

Der Nimbus Client ist für die Dateiübertragung der Images zum Service Node zuständig. Dies geschieht über Java Sockets. Der Client ist auserdem für die Steuerung der Virtuellen Maschinen zuständig.

Mit wget kann der Client von der Nimbus Seite geladen werden.

```
wget http://www.nimbusproject.org/downloads/nimbus-cloud-client-019.tar.gz
```

danach wird er entpackt.

```
tar xfz nimbus-cloud-client-019.tar.gz
```

Damit der Client alle nötigen Einstellungen des Services bekommt wird die generierte Datei, diese Informationen über den Service enthält, in das Client Verzeichnis kopiert.

```
nimbus@ubuntu:/tmp$ cp /tmp/nimbususer/cloud.properties
/home/nimbus/nimbus-cloud-client-019/conf/
```

ein Ordner nimbus wird erstellt in dem verbindungspezifische Einstellungen kopiert werden.

```
mkdir ~/.nimbus/
cp /tmp/nimbususer/*.pem ~/.nimbus/
```

Da der Nimbus Client über Zertifikate mit den Service Node kommuniziert, um sich authentifizieren zu können, müssen die zuvor erstellten Zertifikate in das Client Verzeichnis kopiert werden:

```
cp ./NIMBUS_HOME/var/ca/trusted-certs/*
nimbus-cloud-client-019/lib/certs/
```

Falls der Client auf einer anderer Maschine installiert ist müssen die Daten mit scp oder similar kopiert werden. Sonst sind keine anderen Änderungen nötig bei einer Client Installation auf einem anderen Rechner. Im Falle des Testprojektes wurde der Client erster auf dem Service Node und später dann auf dem VMM Node installiert, da der VMM Host nativ installiert wurde und der Service Node in einer VM läuft.

3.4 Nimbus Client Test

Die Ausführbare Datei für die Kommunikation zwischen Client und Service Node liegt in Client Verzeichnis unter /bin und nennt sich Nimbus-client.sh. Dieses Skript enthält alle möglichen Ausführungen zur Benutzerinteraktion mit dem Service Node.

Mit dem Befehl `cloud-client.sh --status` kann der aktuelle Status der VMs abgefragt werden. Dabei wird nur ein Status angezeigt, falls eine VM sich in einem Zustand befindet (sich auf dem VMM Node in ausführendem Zustand befindet). VMs befinden sich nicht in einem Zustand wenn sie im Columus Space liegen.

```
./bin/cloud-client.sh --status

Querying for ALL instances.

There's nothing running on this cloud that you own.
```

Um die vorhandenen VM Images ab zu fragen, die Im Cumulus gespeichert sind (diese befinden sich auf dem Service Node), wird der Tag `--list` verwendet

```
cloud-client.sh --list

No files.
```

4 Netzwerkkonfiguration

4.1 Allgemein

Da es oftmals vorgekommen ist die Netzwegeinstellunken zu ändern, ob Service Node, VMM Node oder zu Testzwecken, wird hier kurz auf Konfigurationen des Netzwerks eingegangen: Ubuntu, Debian und verwandte Betriebssysteme verwendet eine Script basierte Lösung zur Verwaltung von Netzwerk Einstellungen. Diese Lösung ist bei allen Linux Distributionen gleich. Die Datei für die Einstellungen sind unter /etc/network/interfaces und für DNS unter /etc/resolv.conf (Falls der Netzwerkmanager läuft überschreibt er oftmals

die Datei). Doch falls ein Netzwerk Manager installiert wurde (üblicherweise bei Desktop Umgebungen) so wird die Skript spezifischen Einstellungen teilweise ignoriert, überschrieben oder es kommt zu Konflikten. Die Netzwerk Verwaltung ist bei verschiedenen Linux Distributionen über einen Netzwerk Manager unterschiedlich. Die Vorteile eines Netzwerk Managers bestehen in der Verwaltung von Drahtlosen Verbindungen (aufgrund Erstellung von Verbindungseinstellungen und Passwort Einstellungen) oder Konfigurationen über die Desktop Oberfläche, VPN etc. Die Nachteile des Netzwerks Managers sind, dass dieser keinen Bridge Modus unterstützt und dieser wichtig ist für den VMM.

4.2 Nimbus Netzwerkkonfiguration

Nimbus arbeitet zur Verbindung auch über Hostnamen, deswegen ist es wichtig, dass ein DNS Server existiert der für die Namensauflösung zuständig ist. Jede VM benötigt eine IP Adresse und einen Host Namen diese der VM im Status run zugewiesen werden. Dazu ist es notwendig einen DHCP Server zu konfigurieren, welcher IP Adressen zu weißt. In Linux gibt es unterschiedliche DHCP Server, dabei ist es nicht wichtig, welcher DHCP Server verwendet wird. Wichtig ist allerdings, dass die DHCP Einstellungen sich mit den Netzreservierungskonfigurations Dateien vom Nimbus Service decken. Jede Virtuelle Maschine besitzt ein Public und ein Private Netzwerk. Das Public Netzwerk ist der Zugriffspunkt über den der Client auf die Virtuelle Maschine zugreift. Dabei wird ein DNS Server benötigt und eine Liste von Verfügbaren Netzwerk Slots. Die Datei zur Konfiguration befindet sich unter

```
$NIMBUS_HOME/services/etc/nimbus/workspace-service/network-pools/.
```

Die Public Netzwerk Datei wird nun auf die Bestehende Netzwerkstruktur angepasst. Ein DHCP Server und ein Gateway (Windows 2008) besteht bereits unter der IP Adresse 192.168.10.2.

Die Konfiguration der Netzwerkslots sieht folgende Syntax vor: hostname ipaddress gateway broadcast subnetmask [MAC] Folgende Slots werden im Public Netzwerk reserviert:

```
pub01 192.168.10.100 192.168.10.2 192.168.10.255 255.255.255.0 none
pub02 192.168.10.101 192.168.10.2 192.168.10.255 255.255.255.0 none
pub03 192.168.10.102 192.168.10.2 192.168.10.255 255.255.255.0 none
pub04 192.168.10.103 192.168.10.2 192.168.10.255 255.255.255.0 none
pub05 192.168.10.104 192.168.10.2 192.168.10.255 255.255.255.0 none
pub06 192.168.10.105 192.168.10.2 192.168.10.255 255.255.255.0 none
pub07 192.168.10.106 192.168.10.2 192.168.10.255 255.255.255.0 none
pub08 192.168.10.107 192.168.10.2 192.168.10.255 255.255.255.0 none
```

Private Netzwerk:

```
# DNS IP address (or 'none'):
192.168.10.2
# hostname ipaddress gateway broadcast subnetmask [MAC]
priv001 192.168.10.98 192.168.10.2 192.168.10.255 255.255.255.0
priv002 192.168.10.99 192.168.10.2 192.168.10.255 255.255.255.0
```

Nach Fertigstellung der Netzwerk Einstellungen wird der Service neu gestartet

nimbusctl services restart Unter dem Ordner

```
$NIMBUS_HOME/services/var/nimbus/
```

kann man sich in den Dateien

```
dhcp.entries,ip_macs.txt,control.netsample.txt
```

die DHCP zugewiesenen Adressen und generierten Mac Adressen anschauen und noch andere Netzwerkspezifische Zuweisungen.

Bezüglich der Portabilität des Laptops für die anschließende Präsentation wird doch ein eigener DHCP Server installiert.

Dazu werden erst einmal die Netzwerkeinstellungen geändert:

```
IP 192.168.10.10
SUB 255.255.255.0
Standartgateway 192.168.10.2
```

Zusätzlich wird die Datei /etc/resolv.conf geändert und der DNS Server eingetragen.

4.3 Nimbus Netzwerkkonfiguration

Der isc-dhcp Server wird mit folgender Konfiguration installiert:

```
#############################################################################
#
# /etc/dhcp/dhcpd.conf
# Konfigurationsfile vom DHCP-Server (isc-dhcp-server)
#
# 2011-08-28; Emanuel Duss; Erste Version
#
#############################################################################

#############################################################################
# DHCP-Server
#############################################################################
```

```
# Authoritative = Aktiv (Sendet DHCPNAK)
authoritative;

# Logging
log-facility local7;

# Lease-Time (86400 = 1 Tag)
default-lease-time 86400;
max-lease-time 86400;


#############################################################################
# Globale Optionen
#############################################################################

# Domain
option domain-name "muhlan.home";

# DNS-Server
option domain-name-servers 192.168.10.2;

# NTP-Server
option ntp-servers 192.168.10.2;


#############################################################################
# Subnetz
#############################################################################

# Subnetz 10.0.0.0/24
subnet 192.168.10.0 netmask 255.255.255.0 {
  # Range
  range 192.168.10.2 192.168.10.254;
  # Default-Gateway
  option routers 192.168.10.2;
}


#############################################################################
# Hosts
#############################################################################

# Statische Adressen
```

```
# EOF
```

und schließlich gestartet

```
/etc/init.d/isc-dhcp-server start
```

5 VMM Node

5.1 XEN

Xen ist ein Hypervisor. Eine Software welche den Betrieb mehrerer Betriebssysteme auf einem Host erlaubt. Dabei läuft XEN direkt auf der Hardware was einen geringen Overhead erbringt. Die einzelnen Gast Systeme nennt man Domänen. Dabei verwendet XEN Virtuellen Speicher dieser er den einzelnen Domänen zuweist. Die erste Domäne hat eine besondere Rolle. Sie dient der Interaktion mit dem eigentlichem Hypervisor. Diese Domäne ist in der Lage andere Domänen zu starten, stoppen und zu verwalten. Die erste Domäne muss deswegen auf Dom0 laufen des Privilegsmodus der CPU diese auch Dom U genannt wird. Um völlige Transparenz zu garantieren ist es erforderlich, dass die CPU bestimmt Unterstützungen für Virtualisierungstechniken bietet. Dadurch müssen die Betriebssysteme, diese in den Domänen laufen, nicht angepasst werden, also können auf ihrem ursprünglichen Ring laufen.

5.2 KVM

Kernel based Virtual Machine (KVM) ist eine Linux Kernel Infrastruktur für Virtualisierungen. Dabei verwendet es Hardware Virtualisierungstechniken der Prozessoren. Bestandteil somd die Kernel Modul kvm.ko und zusätzliche Hardwarespezifische Module für Intel CPUs oder AMD Cpus. KVM stellt eine Infrastruktur bereit zur Emulation von Hardware. Ein modifiziertes QEMU ist momentan die einzige Möglichkeit dieses zu nutzen.

5.3 Umsetzung der Virtualisierungslösung

Da Xen per default gesetzt ist in NImbus wurde zuerst versucht die Virtualisierung über XEN zum laufen zu bringen. Folgende Betriebssysteme wurden getestet mit XEN: Ubuntu 12, Ubuntu 10, Debian 6, SuSE Server, Fedora 12.

Es war nicht möglich XEN zum laufen zu bekommen außer unter SUSE. Allerdings war es nicht unter SuSE umsetzbar die Nimbus Infrastructure zu installieren. XEN ist im Gegensatz zu KVM kein Bestandteil des Linux

Kernels und somit muss XEN als Kernel gebootet werden. Unter Ubuntu 12 und Debian war es nicht möglich den Hypervisor Monitor zu starten. Dabei wurde ein vor kompilierter Kernel verwendet. Bei Ubuntu 10 wurde der Kernel selbst kompiliert was allerdings auch zu einer Kernel Panik geführt hatte. Fedora und Mint Linux trugen in Grub den Kernel gar nicht ein und es war nicht möglich diesen manuell ein zu tragen. Getestet wurde das ganze auf 2 unterschiedlichen Rechnern mit einem Intel 7300 (Laptop) und einem Phenom 2 diese laut der Spezifikationen XEN unterstützen. Auf einem 3ten Rechner (AMD FX) wurde XEN zu einem späteren Zeitpunkt mit genau dem selben Installationsvorgang zum laufen gebracht ohne jegliche Probleme. Aufgrund nicht spezifizier-barer Ergebnisse lässt sich vage darauf schließen, dass die XEN Installationen fehl schlugen aufgrund Treiber Problemen. Aus diesem Grund wurde die Virtualisierung mit KVM gemacht.

5.4 KVM Installation

Damit KVM im Linux Kernel aktiviert wird werden folgende Pakete benötigt: qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils Hier ist zu erkennen, dass das Paket libvirt mit installiert wird, dieses das Tool virsh mitliefert, mit dem auf den Hypervisor Monitor zugegriffen werden kann. Zu beachten ist hierbei das der normale Benutzername nimbus in die Gruppe libvirtd mit auf genommen wird, damit der Benutzer die Berechtigung hat virsh zu benutzen und auf den Hypervisor zu zu greifen. Da von jeder VM der Zugriff vom Netzwerk als NAT konfiguriert ist muss der VMM Node eine Bridge haben. Dies bedeutet, dass alle Netzwerkzugriffe über diese Bridge gehen. Dazu wird die Interface Datei folgend geändert:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet manual

auto virbr0
iface virbr0 inet static
        address 192.168.10.10
        network 192.168.10.0
        netmask 255.255.255.0
        broadcast 192.168.10.255
        gateway 192.168.10.2
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

Nach dem Neustart des Netzwerkes werden die Verbindungen getestet.

5.5 Installation VMM Nimbus Software

UM die ganzen Scripte für die Konfiguration zu testen wird zuerst ein VMM installiert und mit Hilfsskripten getestet. Als nächster Schritt wird die Nimbus VMM Software Workspace control installiert. Workspace control beinhaltet Scripte diese zum Testen helfen ob die XEN/KVM/libvirt Installation richtig funktioniert mit den programmierten Zugriffs Pattern.

Bevor ein VMM Node mit Nimbus Software automatisiert wird, muss ein VMM Node erstellt werden. Dazu muss zuerst überprüft werden ob alle Virtualisierungen und Netzwerk Abhängigkeiten vollständig vorhanden sind. Der Workspace-Control verwaltet XEN oder KVM Virtuelle Maschinen. Diese verwenden eine Bibliotheken welche sich libvirt nennt.

Mit dem Befehl

```
virsh -c 'xen+unix:///' list
```

kann überprüft werden, ob Zugriff auf den Hypervisor besteht. Dies gelingt zwar dadurch, dass der Benutzer explizite Berechtigung hat anhand der Gruppe libvirtd, aber es müssen noch zusätzliche Einstellungen vorgenommen werden da sonst später Schwierigkeiten entstehen. Hierzu wird die Datei

```
/etc/libvirt/qemu.conf
```

geöffnet und folgende Einträge gesetzt:

```
user = "nimbus"
group = "nimbus"
dynamic_ownership = 0
```

In der Datei

```
/etc/libvirt/libvirtd.conf
```

werden Einträge ergänzt, damit der Benutzer nimbus auf den Socket von libvirt schreiben kann und nicht nur lesen:

```
unix_sock_group = "libvirtd"
unix_sock_ro_perms = "0770"
unix_sock_rw_perms = "0770"
unix_sock_dir = "/var/run/libvirt"
```

Workspace-control selbst arbeitet im root Modus, damit aber die Virtuellen Maschinen vom Client konfiguriert werden können müssen die Zugriffe

über virsh zum Hypervisor ohne Eingabe eines Passworts oder Sudo Mode funktionieren.Auf der Seite von Nimbus kann Workspace-control heruntergeladen werden und entpackt werden. Unter /opt wird dann ein Verzeichnis mit dem Namen nimbus angelegt und das entpackte workspace-control Verzeichnis dort hin kopiert. Um zu überprüfen ob die Ordnerstrukturen angelegt sind kann folgender Befehl verwendet werden:

```
root # [ -f /opt/nimbus/bin/workspace-control.sh ] && echo "Correct."
```

Damit der Benutzer nimbus auf auf spezifische Teil von workspace-control zugreifen kann werden Änderungen vorgenommen:

```
root # cd /opt/nimbus
root # chown -R root bin etc lib libexec src
root # chown -R nimbus var
root # find . -type d -exec chmod 775 {} \;
root # find . -type f -exec chmod 664 {} \;
root # find bin sbin libexec -iname "*sh" -exec chmod 755 {} \;
```

5.6 Testen der Basis Abhängigkeiten

Bevor die das testen der Viruellen Maschonen beginnt müssen grundlegende Voraussetzungen da sein. Das workspace-control Scripte macht eine Annahme über das bestehende System. Nun wird die Basis Abhängigkeit getestet. Dazu existiert ein Ordner sbin dieser sich unter /opt/nimbus befindet.

```
./sbin/test-dependencies.sh
Python 2.7.3 (default, Apr 13 2012, 20:15:24)
| [GCC 4.6.3 20120306 (Red Hat 4.6.3-2)]

OK, looks like the Python dependencies are set up.
```

Damit KVM getestet werden kann und die Konfigurationen von workspace-control und die Zugriffsberechtigungen, kann unter der Nimbus Download Seite ein Test Image für KVM heruntergeladen werden. Bevor dieses aber gestartet werden kann, müssen noch spezifische Konfigurationen für KVM geändert werden. Mit dem Befehl brctl show werden die Netzwerk interfaces angezeigt. Dabei spielt der Name eine Rolle der zuvor konfigurierten Bridge. Dieser wird nun unter /opt/nimbus/etc/workspace-control/networks.conf als default bridge eingetragen. Auch muss in dieser Datei das Script von XEN zu KVM geändert werden: xen-ebtables-config.sh nach kvm-ebtables-config.sh. Da nimbus der user ist, dieser mit dem Hypervisor und workspace-control kommuniziert müssen spezifische Scripts lesbar sein für den Benutzer nimbus ohne Eingabe eines Passwortes, sonst würde die Automatisierung nicht funktionieren. Mit dem Befehl

```
/usr/bin/sudo /opt/nimbus/libexec/workspace-control/kvm-ebtables-config.sh
```

kann überprüft werden, ob nimbus die Datei als sudo ausführen kann ohne Passwort. Dazu muss in der visudo Datei (am besten weit unten) entsprechende Freigaben eingetragen werden:

```
nimbus ALL=(ALL) NOPASSWD: /opt/nimbus/libexec/workspace-control/mount-alter.sh
nimbus ALL=(ALL) NOPASSWD: /opt/nimbus/libexec/workspace-control/ kvm-ebtables-config.sh
```

Dadurch kann nun der Benutzer nimbus ohne die Eingabe eines Passwortes auf die Dateien zugreifen.

5.7 Testen der Virtuellen Maschinen

durch starten des Service Nodes wurde eine Text Datei erstellt diese sich unter

```
$NIMBUS_HOME/services/var/nimbus/control.netsample.txt
```

befindet. Diese Datei enthält spezifische Knetwerkinformationen die libvirt benötigt um die XML Datei zu generieren, diese die Einstellungen für die Virtuelle Maschine hat. Im sbin verzeichnis- von workspace-control kann mit folgendem Befehl die XML File für die Virtuelle Maschine erstellt werden:

```
./sbin/libvirt-xml.sh --image /tmp/nimbus-z2c --netsample \
/tmp/control.netsample.txt --memory 256 --mountpoint hda > /tmp/z2c.xml
```

Dabei ist netsample der Verweis zu der vom Service Node konfigurierten Datei für Netzwerk Einstellungen und Image der Ort des geladenen KVM Images. Die XML Datei wird dann unter /tmp/z2c.xml gespeichert. Beim erstellen der XML Datei erscheint jedes mal eine Fehlermeldung, dass die Domäne test-control nicht erstellt werden kann. Dies ist ein Name welcher in der Datei libvirt-xml.sh gespeichert ist. Da die Ursachen unbekannt waren für den Fehler wurde die Workspace-control noch auf anderen Betriebssystemen installiert mit dem selben Ergebnis. Durch nachfragen in der E-mail Verteiler Liste von Nimbus wurde klar, dass diese Meldung einfach ignoriert werden kann. Bis dahin hat aber das Problem eine Menge Zeit benötigt. Wenn eine Virtuelle Maschine im Hypervisor läuft und diese wieder zerstört werden muss, bietet nimbus ein Script an, dieses sich ./sbin/destroy-control-test.sh nennt. Ohne ausführen dieses Scripts kann keine weitere Virtuelle Maschine geladen werden. Um die VM zu testen kann mit

```
virsh -c 'qemu:///system' create /tmp/z2c.xml
```

die XML Datei geladen werden und gemäß den Definitionen der XML Datei die VM gestartet werden. Es konnte nicht über die Konsole auf die Virtuelle

Maschine zugegriffen werden, wie in der Nimbus Dokumentation beschrieben, da die XML Datei dies nicht definiert hat. Bei dem Versuch das geladene Image von der Nimbus Seite mit virsh zu laden war keine Verbindung möglich und die Virtuelle Maschine war nicht pingbar. Beim manuellen laden (ohne die nimbus XML Datei) passierte nach laden von GRUB nichts mehr und die CPU war lediglich auf 100 Prozent die ganze Zeit. Aus diesem Grund wurde eigene Virtuellen Maschinen installiert über virsh. Problematisch wurde es dabei die XML Konfigurationen von Nimbus nach zu bauen, da sonst der Wechsle von eigener XML Datei zur workspace-control generierten XML Datei Probleme brachte. Dies äußerte sich durch Fehlkonfigurationen der Netzwerkadapter der VMs, Netzwerk Interfaces waren nicht mehr in der Lage zu starten oder waren nicht auffindbar etc. Zum zerstören vom Einbinden der Images bietet NImbus ein Script dieses im Workspace-control Ordner liegt:

```
./sbin/destroy-control-test.sh
```

Für Überprüfungen bietet Workspace-control eine generierte Log Datei diese unter

```
./sbin/cat-latest-logfile.sh
```

zu finden ist.

6 Einbinden vom VVM Node zum Service

6.1 SSH Einstellungen

Damit der Service Node mit dem VMM Node automatisch kommunizieren kann ist eine passwortfreie Verbindung notwendig. Dies wird über einen private und public Keys umgesetzt . Damit die Kommunikation beidseitig funktioniert müssen 2 Schlüsselpaare erstellt werden und diese ausgetauscht werden. Vom Service Node zum VMM Node:

```
nimbus@service $ ssh-keygen
nimbus@service $ scp ~/.ssh/id_rsa.pub nimbus@nimbus-Latitude-D630:service_rsa.pub
nimbus@nimbus-Latitude-D630 $ cat service_rsa.pub >> .ssh/authorized_keys
```

Analog vom VMM Node zum Service Node:

```
nimbus@nimbus-Latitude-D630 $ ssh-keygen
nimbus@service $ scp nimbus@nimbus-Latitude-D630:~/.ssh/id_rsa.pub ./vmm1_rsa.pub
nimbus@service $ cat ./vmm1_rsa.pub >> .ssh/authorized_keys
```

6.2 Integration des VMM Nodes im Service

Da dem NImbus Service der VMM Node noch vollkommen unbekannt ist, gibt es ein initzialisierungsscript auf dem Service Node unter

```
./bin/nimbus-configure --autoconfig
```

Dieses Installation Script testet zuerst die konfigurierten Einstellungen. Danach wird der VMM Node angegeben und der gegenseitige Zugriff über SSH wird getestet. Auch werden die Strukturen der Dateien und Ordner auf dem VMM Node überprüft. Ist alles erfolgreich abgeschlossen wird der VMM Node im Service integriert und der temporäre Modus vom Service deaktiviert. Es können beliebig viele VMM Nodes dem Service hinzugefügt werden.

7 Testen der Einstellungen

Die erstellen KVM Images müssen einen Ordner unter .ssh unter /root besitzen, da beim integrieren der VM die Keys dort abgelegt werden damit vom Client ein Passwortfreier Zugriff auf die VM möglich ist. Da KVM verwendet wird, muss der Eintrag

```
vws.metadata.mountAs=hda
```

unter conf/cloud.properties beim Client noch ergänzt werden. Der Client hat verschiedene Optionen zur Kommunikation. Die Gesamtkommunikation und die Möglichkeiten, diese der Client besitzt wird über das Script

```
./bin/cloud-client.sh
```

im Client Ordner getätigt. Hierbei gibt es verschiedene Schlüsselwörter: Listet alle VMs auf diese sich im Columus Space befinden:

```
--list
```

Transferiert ein Image einer Virtuellen Maschine zum Columus Space:

```
--transfer --sourcefile <path>
```

Löscht eine VM vom Columus Space:

```
--delete --name <name>
```

kopiert eine VM vom Columus Space zu einem VMM Node und führt sie dort aus. Der VM wird eine IP und einen Host Namen automatisch gegeben:

```
--run --name <name> (optional) --hour <zahl>
```

Zeigt den Status der aktiven VMs auf den VMM Nodes an:

```
--status
```

Kopiert die VM wieder in den Columus Space:

```
--save --handle <handle>
```

Stoppt die VM und löscht diese vom VMM Node:

```
--terminate --handle <handle>
```

8 Fazit

Nimbus Infrastructure ist ein kostenloses Projekt zur private Cloud Lösung. Da dies aber ein Nischenprojekt ist und die Community nicht sehr groß ist, ist im Falle von Fehlern der Aufwand sehr hoch, da sich stark in die Komponenten eingearbeitet werden muss von denen Nimbus abhängig ist. Die Dokumentation selbst war teilweise unvollständig oder Änderungen bei unterschiedlichen Versionen wurden nicht in die Dokumentation mit übernommen. Auf Grund dieser Tatsachen habe ich einen Gesamtaufwand von über 200 Stunden investieren müssen

9 Quellen

www.nimbusproject.org/ wikipedia und noch andere Seiten