

Infrastructure as a Service mit der Cloud Computing Lösung OpenStack auf SUSE Linux Enterprise Server 11

B1 Systems GmbH

Version 2.0 vom 23. September 2011



Dieses Dokument bezieht sich auf SUSE Linux Enterprise Server 11 SP1.

Dieses Dokument untersteht der „Creative Commons Lizenz Namensnennung – Weitergabe unter gleichen Bedingungen 2.0 Deutschland (CC BY-SA 2.0)“ (<http://creativecommons.org/licenses/by-sa/2.0/de/legalcode>) und darf nur unter Einhaltung dieser Lizenz verwendet werden.

Die aktuelle Version dieses Dokuments finden Sie hier:

http://www.b1-systems.de/whitepaper/OpenStack_Cactus_on_SLES11.pdf.

Mit Fragen, Anmerkungen oder um Fehler zu melden wenden Sie sich bitte an doku@b1-systems.de.

B1 Systems ist auf Consulting und Support rund um Linux und Open Source spezialisiert und beschäftigt mehr als 50 Consultants, Entwickler und Trainer. B1 Systems agiert unabhängig und international im Bereich Virtualisierung, Cluster, Cloud Computing und System Management und beteiligt sich aktiv an Open Source Projekten, wie z. B. OpenStack. Neben professioneller Beratung und kundenspezifischen Entwicklungen, rundet ein hochqualifizierter Support die Leistungen von B1 Systems ab.

Bei den folgenden Begriffen handelt es sich um eingetragene Marken:

AMD, AMD Virtualization, AMD-V, DB2, Debian, Intel, Java, Linux, Microsoft Windows, Novell, OpenStack, openSUSE, QEMU, Solaris, SUSE, SUSE Linux, SUSE Linux Enterprise Server, YaST, Xen.

Wir weisen darauf hin, dass alle in diesen Materialien verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen auch ohne besondere Kennzeichnung im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Inhaltsverzeichnis

1	Einführung	1
2	Szenario	6
2.1	Vorbereitung	6
2.1.1	Installation von SLES11 SP1	6
2.1.2	Repository hinzufügen	7
2.2	Abhängigkeiten	8
2.2.1	IP-Forwarding	8
2.2.2	Kernelmodule	8
2.2.3	Sonstiges	9
2.3	Dienste	9
2.3.1	MySQL	9
2.3.2	RabbitMQ	11
2.3.3	libvirt	12
3	OpenStack Image Service - glance	13
3.1	Übersicht	13
3.2	Vorbereitung	13
3.3	Installation	14
3.4	Konfiguration	14
3.5	Start	15
3.6	Images hinzufügen per glance-upload	15
4	OpenStack Compute – nova	19
4.1	Übersicht	19
4.2	Installation	20
4.3	Konfiguration	20
4.3.1	Datenbank	21
4.3.2	AMQP / RabbitMQ	22
4.3.3	Konfiguration der API	23
4.3.4	Konfiguration des Objectstorages	23
4.3.5	Virtualisierungsart	23
4.3.6	Netzwerkkonfiguration	24
4.3.7	Starten der Dienste	24
4.3.8	Logfiles	25
4.4	Kommandübersicht	26
4.5	nova-manage	26
4.5.1	Benutzer	26
4.5.2	Projekte	27
4.5.3	Netzwerke	27
4.5.4	Zertifikate exportieren	27
4.5.5	SSH Key	28
4.5.6	Security Groups	28
4.5.7	service list	29
4.6	Anbindung von glance	29

4.7	Verfügbare Images anzeigen	30
4.7.1	euca2ools	30
4.7.2	nova	30
4.8	Instanzen	31
4.8.1	Starten per euca2ools	31
4.8.2	Starten per nova	32
4.8.3	Stoppen per euca2ools	34
4.8.4	Stoppen per nova	34
4.9	Volumes	34
4.9.1	Vorbereitung	34
4.9.2	Erzeugen und Aktivieren	36
4.9.3	Entfernen	37
4.9.4	Löschen	37
5	OpenStack Object Storage - swift	38
5.1	Übersicht	38
5.1.1	Proxy Server	38
5.1.2	Ringe	39
5.1.3	Object Server	39
5.1.4	Container Server	39
5.1.5	Account Server	40
5.1.6	Replikation	40
5.1.7	Updates	40
5.1.8	Auditors	40
5.2	Vorbereitung	41
5.2.1	Partitionierung	41
5.2.2	rsync	42
5.3	Installation	43
5.4	Konfiguration	43
5.4.1	Proxy Server	43
5.4.2	memcached	44
5.4.3	Account Server	45
5.4.4	Container Server	45
5.4.5	Object Server	46
5.4.6	Ringe aufbauen	46
5.4.7	Der erste Start	48
5.4.8	Authentifizierung	48
5.5	Testen	48
5.6	Clients	50
5.7	Glance für die Nutzung von Swift konfigurieren	50

1 Einführung

Cloud Computing ist ein Konzept, um IT-Infrastruktur, z.B. Datenspeicher oder CPU-Leistung, abstrahiert über ein Netzwerk anzubieten und zur Laufzeit den Bedarf dynamisch an die Bedürfnisse anzupassen. Dabei sind die Ressourcen, die genutzt werden, zu jeder Zeit erweiter- oder reduzierbar.

Unterschieden wird dabei nach privaten und öffentlichen Clouds (engl. *private/public cloud*). Ein Mischbetrieb ist möglich und wird dann als *Hybrid Cloud* bezeichnet. Die private Cloud nutzt man, wenn man eine solche Cloud-Infrastruktur innerhalb des eigenen LANs benutzt. Eine öffentliche Cloud hingegen bedeutet die Nutzung einer Cloud über das Internet bei einem öffentlichen Anbieter, beispielsweise Amazon. Ein Mischbetrieb beschreibt die Kombination dieser beiden.

Es existieren verschiedene Arten von Cloud-Architekturen:

Infrastructure as a Service (IaaS) Bereitstellung einer vollständigen Infrastruktur in virtuellen Maschinen; umfasst CPU, Speicher, Betriebssystem und Netzwerkkonfiguration.

Platform as a Service (PaaS) Hierbei wird eine Plattform für den Benutzer bereitgestellt. Es könnte sich zum Beispiel um eine vollständige IDE zur Java-Entwicklung handeln. Mit dem darunter liegenden Betriebssystem und Netzwerkkonfiguration kommen die Benutzer nicht in Berührung.

Software as a Service (SaaS) SaaS stellt dem Benutzer lediglich eine einzelne Software zur Verfügung. Dabei kann es sich zum Beispiel um eine Webmaileroberfläche handeln.

Über eine Cloud ist es einem Anbieter möglich, seinen Kunden eine Infrastruktur bereitzustellen und die Nutzung der Ressourcen per Benutzer abzurechnen. So würde ein Kunde nur für die benutzte Rechenzeit zahlen und nicht für die Stunden, in denen die Maschinen nicht genutzt werden. Auch gibt es die Möglichkeit einer Skalierung, so können ab einem bestimmten Auslastungsgrad weitere virtuelle Maschinen gestartet und somit die Last verteilt werden.

OpenStack bildet die *Infrastructure as a Service*-Architektur ab. Es besteht aus mehreren Komponenten, *Compute*, *Object Storage* und *Glance*. Weitere Komponenten sind in Entwicklung. *Compute* ist der Teil, der für das Verwalten und Einrichten größerer Gruppen von virtuellen Maschinen zuständig ist. Der *Object Storage* hingegen ist für das redundante Speichern von Daten verantwortlich. Da die Daten verteilt werden, spricht man vom *Infinite Storage*. Der *Object Storage* ist vergleichbar mit Amazons S3.

2005 entwickelte die Firma *Rackspace* die *Rackspace Cloud* und entschloss sich 2009, die Software neu zu schreiben. Im März 2010 wurde diese dann als Open Source freigegeben (dies betrifft den Storage-Teil). Im Mai 2010 gab die NASA ihr *Nebula*-Projekt frei und bereits im Juni schlossen sich die beiden Projekte zu *OpenStack* zusammen. Im Juli 2010 fand das erste Zusammentreffen der Entwickler mit einer Zielvorgabe statt, und schon im Oktober desselben Jahres wurde das erste Release namens *Austin* veröffentlicht. Daran waren bereits 35 Partnerfirmen beteiligt. Das nächste Release *Bexar* wurde im Februar

2011 veröffentlicht, danach folgte die Version mit dem Namen *Cactus*, freigegeben am 15. April 2011. Aktuell ist *Diablo*, erschienen am 22. September 2011.

Die folgenden Seiten befassen sich mit der Installation von *OpenStack* unter SUSE Linux Enterprise Server 11. Die einzelnen Komponenten von *OpenStack* werden in den folgenden Kapiteln jeweils separat, aber aufeinander aufbauend erklärt.



brontes
nova-compute



ares
nova-api
nova-scheduler
nova-network



deimos
MySQL
iSCSI
RabbitMQ



chronos
glance-api
glance-registry

Die in dieser Anleitung verwendeten Systeme sind auf dem oberen Bild dargestellt. Unter den einzelnen Hosts stehen deren Aufgaben.

Systeme

Name	IP-Adresse	Funktion
ares	192.168.2.110	<i>Nova (API)</i>
brontes	192.168.2.120	<i>Nova (Compute)</i>
chronos	192.168.2.130	<i>Swift (API), Glance (API, Registry)</i>
deimos	192.168.2.140	<i>Dienste (MySQL , iSCSI , RabbitMQ)</i>

Die einzelnen Komponenten von *Nova* haben unterschiedliche Aufgaben:

Aufgaben

Komponente	Aufgabe
nova-api	kontrolliert den Hypervisor, Storage und das Netzwerk. Als Endpunkt dient die API Authentifizierung, Kommandos und Kontrollfunktionen
nova-compute	startet und stoppt Instanzen, hängt Volumes in die Instanzen ein oder aus
nova-network	verwaltet die Netzadressen, konfiguriert VLANs
nova-scheduler	verteilt die zu startenden Instanzen auf die compute-Nodes
MySQL	verwaltet die Benutzerdaten, Projekte und Netzwerkdaten
RabbitMQ	stellt Messaging Queues zur Verfügung, über die die einzelnen <i>Nova</i> - Komponenten miteinander kommunizieren
glance	verwaltet die Images für die Instanzen

Die folgende Tabelle bietet einen Überblick über die in den Repositories verfügbaren Pakete zu *OpenStack* und deren Funktionen. Zusätzlich zu den von *Nova* benötigten Komponenten sind die für den Objectstore *Swift* existierenden Pakete mit aufgeführt.

Paketübersicht

Paket	Komponente	Beschreibung
openstack-glance	<i>Nova</i>	Dienst zum Registrieren von Images
openstack-nova	<i>Nova</i>	gemeinsame Dateien
openstack-nova-api	<i>Nova</i>	der API-Daemon
openstack-nova-compute	<i>Nova</i>	startet virtuelle Maschinen
openstack-nova-doc	<i>Nova</i>	Dokumentation
openstack-nova-network	<i>Nova</i>	Netzwerkverwaltung
openstack-nova-objectstore	<i>Nova</i>	einfacher Store für Images
openstack-nova-scheduler	<i>Nova</i>	bestimmt, welche Node welche Maschinen startet
openstack-nova-volume	<i>Nova</i>	verwaltet persistente Volumes
openstack-swift	<i>Swift</i>	<i>Swift</i> zweiter Teil von <i>OpenStack</i> , der große Objectstore
openstack-swift-account	<i>Swift</i>	Liste Container
openstack-swift-auth	<i>Swift</i>	Authentifizierung
openstack-swift-container	<i>Swift</i>	Liste Objekte
openstack-swift-doc	<i>Swift</i>	Dokumentation
openstack-swift-object	<i>Swift</i>	Object Server
openstack-swift-proxy	<i>Swift</i>	API-Endpunkt für <i>Swift</i>
openstack-swauth	<i>Swift</i>	Authentifizierung für <i>Swift</i>
openstack-glance	<i>Glance</i>	<i>Glance</i>
openstack-keystone	<i>Keystone</i>	Identitätsmanagement

Dazu kommen weitere Anwendungen, die zum Betrieb von *OpenStack* benötigt werden:

Übersicht zusätzlich benötigter Pakete

Paket	benötigt für	Beschreibung
MySQL	<i>Nova</i>	Datenbankserver, enthält die Projekte, Benutzer etc.
RabbitMQ	<i>Nova</i>	AMQP Server (Kommunikation der Dienste untereinander)
memcached	<i>Swift</i>	Speicheroptimierung

Weiterführende Informationen finden Sie auf den Webseiten der Projekte:

- <http://www.openstack.org>
- <http://docs.openstack.org>
- <http://www.rabbitmq.org>

Über den IRC-Channel #openstack im IRC-Netzwerk *Freenode* erfahren Sie Unterstützung direkt von den Entwicklern und der Community.

2 Szenario

2.1 Vorbereitung

2.1.1 Installation von SLES11 SP1

Bei der Grundinstallation des Systems wird eine *root*-Partition mit einer Größe von 15 GB angelegt, zusätzlich wird die empfohlene *swap*-Partition erzeugt:

```
# fdisk -l /dev/sda
```

```
Disk /dev/sda: 200.0 GB, 200049647616 bytes
255 heads, 63 sectors/track, 24321 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x0003d4b0
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1	*	1	1958	15727603+	83	Linux
/dev/sda2		1959	3042	8707230	82	Linux swap / Solaris

Installiert wird ein minimales System, dass heißt lediglich mit den Patterns *Minimal* und *base*. Der verbleibende lokale Speicherplatz wird später zum Aufbau des *Distributed Storage* sowie des *Object Storage* verwendet.

Deaktivieren Sie nicht notwendige Dienste. Insbesondere muss *SuSEfirewall2* deaktiviert werden, da *OpenStack* eigene *iptables*-Regeln setzt. Unter SLES11 SP1 funktioniert dies am einfachsten durch folgende Aufrufe:

```
# chkconfig SuSEfirewall2_setup off
# chkconfig SuSEfirewall2_init off
# rcSuSEfirewall2 stop
```

Hier folgt noch eine Übersicht über die aktivierten Dienste nach der Installation des Betriebssystems:

```
# chkconfig | grep " on"
acpid                on
cron                 on
dbus                 on
earlysyslog          on
fbset                 on
haldaemon            on
irq_balancer          on
kbd                  on
microcode.ctl         on
network              on
network-remotefs     on
ntp                  on
random               on
splash               on
splash_early         on
sshd                 on
syslog               on
```

2.1.2 Repository hinzufügen

Zur Verwendung der Pakete benötigen Sie folgende zusätzlichen Quellen, um *OpenStack* installieren und verwenden zu können:

```
http://download.opensuse.org/repositories/isv:/B1-Systems:/devel:/OpenStack/SLE_11_SP1/isv:B1-Systems:devel:OpenStack.repo
```

Konfigurieren Sie diese Repositories auf jedem System, das *Nova* nutzen soll:

```
# zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems:/
  devel:/OpenStack/SLE_11_SP1/isv:B1-Systems:devel:OpenStack.repo
Adding repository 'OpenStack Packages for openSUSE and SLES11 (
  SLE_11_SP1)' [done]
Repository 'OpenStack Packages for openSUSE and SLES11 (SLE_11_SP1)'
  successfully added
Enabled: Yes
Autorefresh: No
URI: http://download.opensuse.org/repositories/isv:/B1-Systems:/
  OpenStack/SLE_11_SP1/

# zypper ref
New repository or package signing key received:
Key ID: C0CB79B9CE9EAAB5
Key Name: isv:B1-Systems OBS Project <isv:B1-Systems@build.opensuse.org
  >
Key Fingerprint: A3A19AC6B7B8D887BA3C46DFC0CB79B9CE9EAAB5
Repository: OpenStack Packages for openSUSE and SLES11 (SLE_11_SP1)

Do you want to reject the key, trust temporarily, or trust always? [r/t
  /a/?] (r): a
Retrieving repository 'OpenStack Packages for openSUSE and SLES11 (
  SLE_11_SP1)' metadata [done]
Building repository 'OpenStack Packages for openSUSE and SLES11 (
  SLE_11_SP1)' cache [done]
All repositories have been refreshed.
```

Hinweis: Standarddienste

Wenn Sie ein eigenes System für notwendige zusätzliche Dienste, z.B. *MySQL*, verwenden, ist es nicht nötig, das Repository vom OpenSUSE Build Service (OBS) einzubinden, wenn die Dienste bereits in SLES11 SP1 enthalten sind. Da im vorgestellten Setup alle Dienste auf einem System betrieben werden und beispielsweise *RabbitMQ* kein Bestandteil von SLES11 SP1 ist, muss auch auf *deimos* dieses Repository hinzugefügt werden.

Hinweis: OpenSUSE Build Service (OBS)

Stefan "seife" Seyfried hat funktionierende RPM-Pakete zusammengestellt. Diese wurden von uns lange getestet und aktualisiert. Sie stehen auf dem OBS im Projekt *ISV:B1-Systems:OpenStack* zur Verfügung.

Hinweis: *Vendor Change*

Da einige Pakete in einer aktuelleren Version als in SLES11 SP1 vorhanden benötigt werden, kann es zu Warnungen bezüglich eines *Vendor Changes* kommen. Diese müssen bestätigt werden, damit die aktuelleren Pakete eingespielt werden können.

2.2 Abhängigkeiten

Für die Verwendung von *Nova* müssen weitere Abhängigkeiten erfüllt werden, damit ein reibungsloser Ablauf gegeben ist.

- IP-Forwarding
- zusätzliche Kernelmodule
- Virtualisierung
- *bridge-utils*

Beachten Sie die einzelnen erforderlichen Abhängigkeiten und die zugehörigen Installationsanweisungen.

2.2.1 IP-Forwarding

Zur Netzwerkkommunikation aktivieren Sie *IP-Forwarding*, damit Pakete über interne Schnittstellen geroutet werden.

Dazu editieren Sie `/etc/sysconfig/sysctl`, damit die Änderung nach einem Neustart des Systems zur Verfügung steht:

```
# Runtime-configurable parameter: forward IP packets.  
# Is this host a router? (yes/no)  
#  
IP_FORWARD="yes"
```

Alternativ schalten Sie *IP-Forwarding* zur Laufzeit ein:

```
sysctl -w net.ipv4.ip_forward=1
```

2.2.2 Kernelmodule

Benötigte Kernelmodule:

- `ext4 - ext4dev-kmp-default`
- `nb`
- `kvm`

Für den Zugriff auf Dateisysteme in den virtuellen Maschinen und die sog. *Config Injection* benötigen Sie unter Umständen *ext4*, um auf das im Image enthaltene Dateisystem zuzugreifen.

Sollte das Modul *ext4* noch nicht zur Verfügung stehen, installieren Sie es:

```
# zypper in ext4dev-kmp-default
Refreshing service 'nu_novell_com'.
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following NEW packages are going to be installed:
  ext4dev-kmp-default ext4dev-kmp-pae kernel-default kernel-default-
    base

The following packages are not supported by their vendor:
  ext4dev-kmp-default ext4dev-kmp-pae

4 new packages to install.
Overall download size: 25.6 MiB. After the operation, additional 69.1
  MiB will be used.
Continue? [y/n/?] (y):
```

Dieses Modul laden Sie unter SLES11 SP1 mit folgendem Kommando:

```
# modprobe --allow-unsupported ext4
```

Das Modul *nbid* sollte zur Verfügung stehen. Darüber werden Images bei der Config-Injection gemountet. Falls das Modul nicht geladen ist, fügen Sie es hinzu:

```
# modprobe nbd
```

Hinweis: Für KVM muss Hardwarevirtualisierung im BIOS aktiviert sein

Zusätzlich zum Modul *kvm* muss das für die Hardwarearchitektur passende Module geladen werden, *kvm_intel* für Intel-VT oder *kvm_amd* für AMD-V.

2.2.3 Sonstiges

```
# zypper install tuncctl bridge-utils kvm virt-utils
[...]
```

Für die Netzwerkkonfiguration auf den *compute-nodes* benötigen Sie Programme zur Verwaltung von Netzwerkbrücken.

2.3 Dienste

2.3.1 MySQL

Die Projektseite von *MySQL* findet sich unter <http://www.mysql.com>.

In der Datenbank werden sämtliche Benutzer, Projekte, Netzwerke, verwendete IP-Adressen und viele weitere Daten gespeichert.

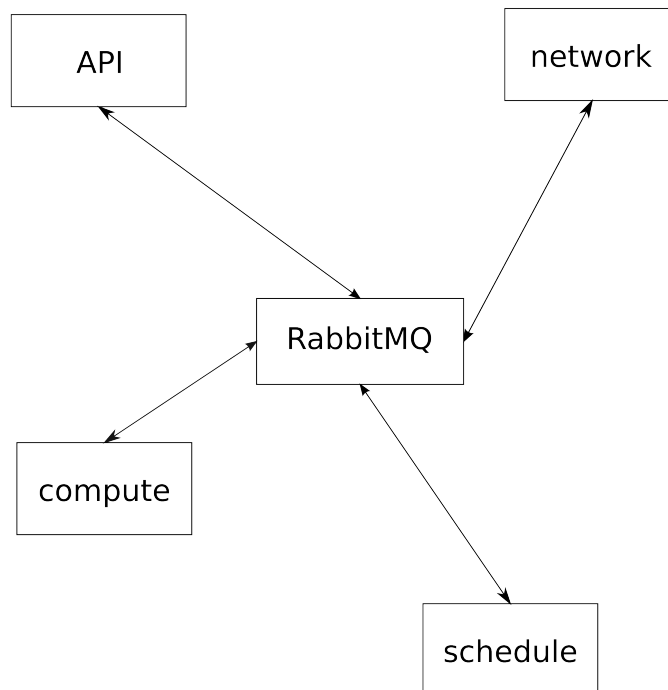
Zuerst installieren und starten Sie den *MySQL* -Server und setzen ein Passwort für den Benutzer *root*. Danach erzeugen Sie eine Datenbank namens *nova* und erlauben dem gerade erzeugten Benutzer den Zugriff auf diese Datenbank. Zuletzt wird *MySQL* so konfiguriert, dass der Server beim Start des Systems aktiviert wird.

```
deimos:~ # zypper install mysql
[...]
deimos:~ # rcmysql start
Creating MySQL privilege database...
[...]
Starting service MySQL                               done
deimos:~ # mysqladmin password testing
deimos:~ # mysql -u root -p
Enter password:
mysql> create database nova;
Query OK, 1 row affected (0.00 sec)
mysql> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'testing
    ' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)
mysql> quit
# chkconfig mysql on
```

2.3.2 RabbitMQ

Die Kommunikation der einzelnen Dienste findet aktuell über das *AMQP* (*Advanced Message Queuing Protocol*) statt. Als Server unterstützt *OpenStack* aktuell *RabbitMQ*.

Wie Sie der folgenden Grafik entnehmen, kommunizieren die einzelnen Komponenten nicht direkt miteinander, sondern nutzen als Vermittler den RabbitMQ-Server:



Fügen Sie das folgende Repository hinzu, um das Paket installieren zu können:

```
# zypper ar http://download.opensuse.org/repositories/network:/  
messaging:/amqp/SLE_11_SP1/network:messaging:amqp.repo  
Adding repository 'Advanced Message Queuing Protocol (AMQP) (SLE_11_SP1  
)' [done]  
Repository 'Advanced Message Queuing Protocol (AMQP) (SLE_11_SP1)'  
successfully added  
Enabled: Yes
```

```

Autorefresh: No
URI: http://download.opensuse.org/repositories/network:messaging:/amqp
    /SLE_11_SP1/

# zypper ref
Repository 'OpenStack Packages for openSUSE and SLES11 (SLE_11_SP1)' is
  up to date.
Retrieving repository 'Advanced Message Queuing Protocol (AMQP) (
  SLE_11_SP1)' metadata [\]

New repository or package signing key received:
Key ID: C59625749EECCAA2
Key Name: network:messaging:amqp OBS Project <network:messaging:
  amqp@build.opensuse.org>
Key Fingerprint: C8A497D3F4008279C357B135C59625749EECCAA2
Repository: Advanced Message Queuing Protocol (AMQP) (SLE_11_SP1)

Do you want to reject the key, trust temporarily, or trust always? [r/t
  /a/?] (r): a
Retrieving repository 'Advanced Message Queuing Protocol (AMQP) (
  SLE_11_SP1)' metadata [done]
Building repository 'Advanced Message Queuing Protocol (AMQP) (
  SLE_11_SP1)' cache [done]
All repositories have been refreshed.

```

Nun installieren Sie das Paket und aktivieren den Server beim Systemstart:

```

deimos:~ # zypper install rabbitmq-server
[...]
deimos:~ # /etc/init.d/rabbitmq-server start
Starting rabbitmq-server: SUCCESS
rabbitmq-server.
# chkconfig rabbitmq-server on

```

Hinweis: Standardkonfiguration lauffähig

Die Konfiguration ist in `/etc/sysconfig/rabbitmq-server` enthalten. In der Standardkonfiguration des SLES11-Paketes sollten keine Anpassungen erforderlich sein.

2.3.3 libvirt

Die Verwaltung der virtuellen Maschinen erfolgt mittels *libvirt*. Dies muss auf den Nodes geschehen, auf denen die Instanzen gestartet werden sollen (*Compute*-Nodes).

```

# zypper install libvirt
[...]
# rm /etc/libvirt/qemu/networks/default.xml

```

Weitere Informationen unter <http://www.libvirt.org>.

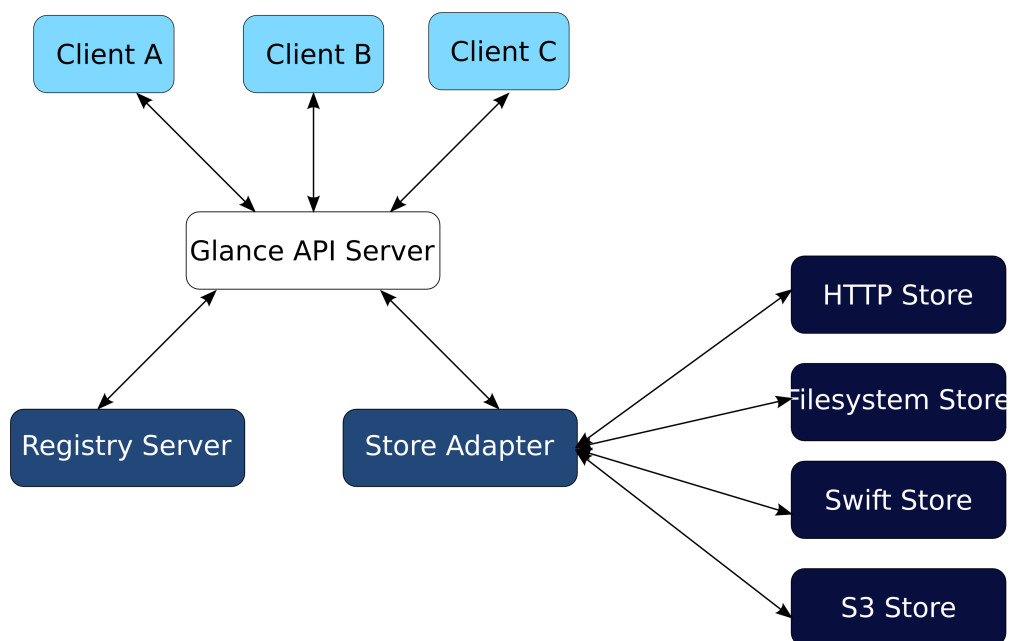
3 OpenStack Image Service - glance

3.1 Übersicht

Glance ist ein Dienst, der für das Entdecken, Registrieren und Empfangen von Imagedateien zuständig ist. Über die API ist es möglich, sowohl die Daten zu empfangen als auch Metadaten abzufragen.

Images, die per *Glance* verfügbar gemacht werden, können an zahlreichen Stellen gespeichert werden, vom einfachen Dateisystem bis zu *Object Stores* wie z.B. *Swift*.

Die Projektseite von *Glance* findet sich unter <http://glance.openstack.org>.



Die einzelnen Clients, also die *Compute*-Nodes, greifen über die API auf den Store Adapter zu. Dieser hat unterschiedliche Möglichkeiten, woher er die benötigten Dateien für die Images bezieht, dazu zählt unter anderem das Dateisystem.

Der folgende Abschnitt erläutert die Konfiguration von *Glance* und die Bestückung mit Images.

3.2 Vorbereitung

Dieses Beispiel nutzt das Dateisystem, um Images dort abzulegen. Alternativ kann *Glance* z.B. auf einen S3-kompatiblen Storage verwendet oder der zum *OpenStack* gehörenden Swift-Store werden.

Zuerst bereiten Sie eine Testpartition oder ein Logical Volume vor und mounten dieses unter `/srv/glance`. Damit diese Partition unter dem benötigten Mountpunkt auch nach einem Neustart des Systems zur Verfügung steht, binden Sie den passenden Eintrag direkt in die `/etc/fstab` ein.

```
chronos:~ # mkdir /srv/glance
chronos:~ # mkfs.xfs /dev/openstack/glance
meta-data=/dev/openstack/glance isize=256      agcount=4, agsize=1966080
          blks
          =                               sectsz=512   attr=2
data      =                               bsize=4096   blocks=7864320, imaxpct
          =25
          =                               sunit=0      swidth=0 blks
naming    =version 2                       bsize=4096   ascii-ci=0
log       =internal log                     bsize=4096   blocks=3840, version=2
          =                               sectsz=512   sunit=0 blks, lazy-count
          =1
realtime  =none                             extsz=4096   blocks=0, rtextents=0
chronos:~ # echo "/dev/openstack/glance /srv/glance xfs noatime,
          nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
chronos:~ # mount /srv/glance
chronos:~ # mount
[...]
/dev/openstack/glance on /srv/glance type xfs (rw,noatime,nodiratime,
          nobarrier,logbufs=8)
```

3.3 Installation

Installiert wird das folgende Paket:

```
chronos:~ # zypper install openstack-glance
```

Zur Nutzung muss *Glance* noch konfiguriert werden.

3.4 Konfiguration

Die Konfiguration von *Glance* findet in zwei Dateien statt, einmal `glance-api.conf`, diese nimmt die Daten für den Speicherort auf, und in der `glance-registry.conf` welche die Datenbankanbindung enthält. Beide Dateien liegen unter `/etc/glance`.

In der Datei `/etc/glance/glance-api.conf` wird folgende Option angepasst:

```
filesystem_store_datadir = /srv/glance
```

Dieses Verzeichnis muss selbst angelegt werden, falls es noch nicht existiert:

```
# mkdir -p /srv/glance
```

Nun benötigt *Glance* noch eine Datenbank für die Speicherung der Imagedaten, hier bietet sich die auch für *Nova* installierte *MySQL* -Datenbank an. Zur Nutzung konfiguriert man die passende Zeile in der Datei `/etc/glance/glance-registry.conf`. Dies überschreibt dann die Verwendung der *sqlite*-Datenbank für *Glance*

```
sql_connection = mysql://root:testing@deimos/glance
```

Wie schon bei *Nova* selbst, wird nun eine *MySQL* -Datenbank erzeugt:

```
deimos:~ # mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 5.0.67 SUSE MySQL RPM

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database glance;
Query OK, 1 row affected (0.00 sec)
mysql> quit
Bye
```

Nun steht eine leere Datenbank für *Glance* zur Verfügung und wird beim ersten Start initialisiert.

3.5 Start

Glance besteht aus zwei Diensten, der API und der Registry. Diese werden gemeinsam im bereitgestellten Startskript gestartet:

```
# glance-control all start
Starting glance-api with /etc/glance/glance.conf
Starting glance-registry with /etc/glance/glance.conf
```

Nun sollten *glance-api* und *glance-registry* laufen, zusätzlich wird von *glance-registry* beim ersten Start die Struktur in der Datenbank angelegt. Sollte der Prozess nicht gestartet werden (Prozesstabelle), ist ein Fehler im Zugriff auf die Datenbank aufgetreten.

Zuletzt wird *Glance* noch per *chkconfig* beim Systemstart aktiviert:

```
# chkconfig openstack-glance on
```

3.6 Images hinzufügen per glance-upload

Auch *Glance* muss natürlich mit Daten bestückt werden, damit den Instanzen Images zur Verfügung gestellt werden können. Dafür gibt es das Kommando *glance-upload*.

Zum Testen des Uploads verwenden Sie das jüngste Image von <http://smoser.brickies.net/ubuntu/ttylinux-uec> und entpacken es:

```
chronos:~ # tar xvzf ttylinux-uec-amd64-12.1_2.6.35-22_1.tar.gz
ttylinux-uec-amd64-12.1_2.6.35-22_1-floppy
ttylinux-uec-amd64-12.1_2.6.35-22_1.img
ttylinux-uec-amd64-12.1_2.6.35-22_1-initrd
ttylinux-uec-amd64-12.1_2.6.35-22_1-loader
ttylinux-uec-amd64-12.1_2.6.35-22_1-vmlinuz
```

Die Schritte zum Hinzufügen zu *Glance* sind ähnlich den Anweisungen für das manuelle Hinzufügen von Images. Nach jedem Schritt müssen Sie sich die *ID* merken, die zurückgemeldet wird.

Hier erfolgt das Hinzufügen des Kernels (der Typ wird übergeben), die Ausgabe verrät, dass die ID 3 vergeben wurde:

```
chronos:~ # glance-upload --type kernel ttylinux-uec-amd64-12.1_2
.6.35-22_1-vmlinuz ttylinux-uec
Stored image. Got identifier: {u'created_at': u'2011-02-10T17:04:03',
u'deleted': False,
u'deleted_at': None,
u'id': 3,
u'is_public': True,
u'location': u'file:///srv/images/3',
u'name': u'ttylinux-uec',
u'properties': {},
u'size': 4404752,
u'status': u'active',
u'type': u'kernel',
u'updated_at': None}
```

Als Nächstes erfolgt das Hochladen der Ramdisk, in diesem Beispiel erhalten Sie die ID 4 zurück.

```
chronos:~ # glance-upload --type ramdisk ttylinux-uec-amd64-12.1_2
.6.35-22_1-initrd ttylinux-uec
Stored image. Got identifier: {u'created_at': u'2011-02-10T17:05:18',
u'deleted': False,
u'deleted_at': None,
u'id': 4,
u'is_public': True,
u'location': u'file:///srv/images/4',
u'name': u'ttylinux-uec',
u'properties': {},
u'size': 5882349,
u'status': u'active',
u'type': u'ramdisk',
u'updated_at': None}
```

Fügen Sie das eigentliche Image hinzu. Damit das Image passend mit Kernel und Ramdisk verknüpft wird, übergeben Sie die in den vorherigen Schritten erhaltenen IDs:

```
chronos:~ # glance-upload --type machine --ramdisk 4 --kernel 3
ttylinux-uec-amd64-12.1_2.6.35-22_1.img ttylinux-uec
Stored image. Got identifier: {u'created_at': u'2011-02-10T17:06:14',
u'deleted': False,
u'deleted_at': None,
```

```

u'id': 5,
u'is_public': True,
u'location': u'file:///srv/images/5',
u'name': u'ttylinux-uec',
u'properties': {u'kernel_id': u'3', u'ramdisk_id': u'4'},
u'size': 25165824,
u'status': u'active',
u'type': u'machine',
u'updated_at': None}

```

Nun ist ein vollständiges Image, bestehend aus Kernel, Ramdisk und dem eigentlichen Abbild registriert und steht den Nodes nach einigen Sekunden zur Verfügung.

Zur Kontrolle, ob die hinzugefügten Daten nun wirklich von *glance* verwaltet werden, achten Sie auf die Ausgabe des folgenden Befehls:

```

# glance index
Found 3 public images...

```

ID	Name	Disk Format
Container Format	Size	
1	None	None
2	None	None
3	None	None

Hier werden nun die gerade über die *Glance* -API hinzugefügten Images tabellarisch aufgelistet. Zusätzlich zu einer ID und dem Namen ist insbesondere die Größe der einzelnen Images interessant. Durch die Nutzung mehrerer Images kann man anhand der Größe eines Images schon vorab die ungefähre Zeit abschätzen, die *Nova* zum Starten einer virtuellen Maschine benötigt.

Weitere mögliche Parameter kann man durch den Aufruf von *glance* sehen, hier wird eine kurze Übersicht und Beschreibung dargestellt:

```

# glance
Usage: glance <command> [options] [args]

Commands:
  help <command>  Output help for one of the commands below
  add              Adds a new image to Glance
  update          Updates an image's metadata in Glance
  delete          Deletes an image from Glance
  index           Return brief information about images in Glance
  details         Return detailed information about images in
                  Glance
  show            Show detailed information about an image in
                  Glance
  clear           Removes all images and metadata from Glance

```

Cache Commands:

```

cache-index           List all images currently cached
cache-invalid         List current invalid cache images
cache-incomplete     List images currently being fetched
cache-prefetching    List images that are being prefetched
cache-prefetch       Pre-fetch an image or list of images into the
                    cache
cache-purge           Purges an image from the cache
cache-clear           Removes all images from the cache
cache-reap-invalid    Reaps any invalid images that were left for
                    debugging purposes
cache-reap-stalled   Reaps any stalled incomplete images

```

Member Commands:

```

image-members        List members an image is shared with
member-images        List images shared with a member
member-add            Grants a member access to an image
member-delete        Revokes a member's access to an image
members-replace      Replaces all membership for an image

```

[...]

Werden von den Optionen weitere Parameter benötigt, so wird dies durch einen Aufruf dargestellt:

```

# glance show
Please specify the image identifier as the
first argument. Example:
$> glance-admin show 12345

```

So erhält man z.B. detaillierte Informationen zum Image mit der ID 5 durch den Aufruf von `glance show 5`:

```

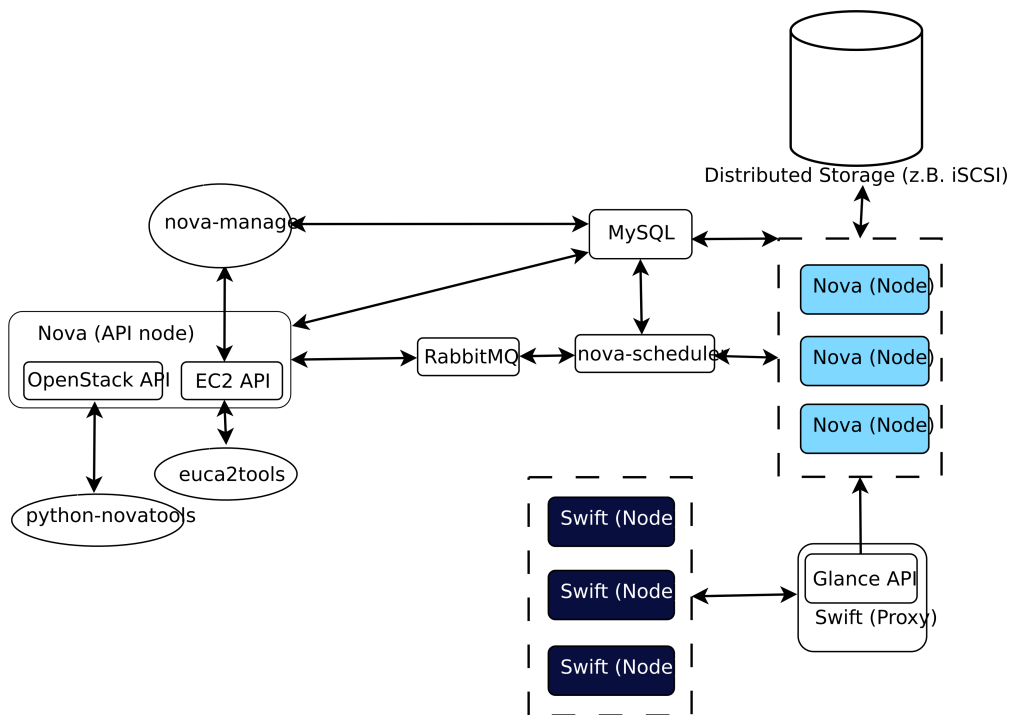
# glance show 5
URI: http://0.0.0.0/images/5
Id: 5
Public: Yes
Name: ttylinux-uec
Status: active
Size: 25165824
Location: file:///srv/glance/images/5
Disk format: None
Container format: None
Property 'kernel_id': 3
Property 'ramdisk_id': 4
Property 'type': machine

```

4 OpenStack Compute – nova

4.1 Übersicht

Zum besseren Verständnis davon, wie die Komponenten von *Nova* zusammenarbeiten, folgendes Bild:



Sie haben zwei Möglichkeiten, Befehle an die API-Node abzusetzen, entweder mit den *Nova*-eigenen Kommandos, diese nutzen dann die *Nova*-API oder alternativ mit den *euca2tools*, diese benutzen die Amazon-API, die aber von *Nova* verstanden wird. Der API-Node kommuniziert dann per RabbitMQ mit dem Scheduler, dieser wiederum nutzt die MySQL-Datenbank für Daten und verwaltet dann die compute-Nodes.

Um die benötigten Projekte, Benutzer und weitere Daten anzulegen, können Sie mit dem Kommandozeilenprogramm *nova-manage* Daten direkt in der MySQL-Datenbank verändern.

Die einzelnen Compute-Nodes können auf einen zur Verfügung gestellten Speicher direkt zugreifen, dieser wird z.B. per iSCSI zur Verfügung gestellt. Für die benötigten Images der virtuellen Maschinen fragen die Compute-Nodes bei *Glance* an. Dieses stellt dann die Images bereit. Diese wiederum können entweder direkt im Dateisystem gespeichert sein oder, wie auf der Grafik dargestellt, im ObjectStorage *Swift* liegen.

4.2 Installation

Auf dem Cloud Controller, nachfolgend als *API-Node* bezeichnet, der auf dem Host *ares* laufen soll, werden Dienste betrieben, die nur einmal benötigt werden. Dies sind `nova-api` sowie `nova-scheduler`. Es empfiehlt sich, für diese Dienste ein eigenständiges System zu verwenden.

API-Node benötigt `nova-api` und `nova-scheduler` :

```
ares:~ # zypper install openstack-nova openstack-nova-api
       openstack-nova-scheduler
```

Die Dienste `nova-compute` und `nova-volume` sowie ggf. `nova-network` werden auf den Systemen benötigt, auf denen später die virtuellen Maschinen betrieben werden sollen und die nachfolgend als *Compute*-Nodes bezeichnet werden. Die Verwaltung der virtuellen Maschine wird mittels der *libvirt* durchgeführt.

Compute-Node benötigt `nova-compute`, `nova-network` und `nova-volume` :

```
brontes:~ # zypper install openstack-nova openstack-nova-compute
          openstack-nova-network openstack-nova-volume
```

4.3 Konfiguration

Damit die einzelnen Hosts direkt über ihre Namen ansprechbar sind und da im Beispiel kein Nameserver zur Verfügung steht, tragen Sie die einzelnen Adressen und Namen direkt in die Datei `/etc/hosts` ein.

Nehmen Sie diese Konfiguration auf jedem der vier Systeme vor.

```
/etc/hosts
192.168.2.110 ares
192.168.2.120 brontes
192.168.2.130 chronos
192.168.2.140 deimos
```

Die Hauptkonfigurationsdatei von *Nova* heißt `nova.conf` und befindet sich im Verzeichnis `/etc/nova`. Die ausgelieferten Pakete enthalten eine blanke Konfiguration mit den nur benötigten Parametern. Diese passen Sie entsprechend an:

```
/etc/nova/nova.conf
--daemonize=1
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--verbose
--sql_connection=mysql://root:<mysql-password>@<IP>/nova
--s3_host=<IP>
--rabbit_host=<IP>
--ec2_url=http://<IP>:8773/services/Cloud
```

```
--network_manager=nova.network.manager.VlanManager
--fixed_range=192.168.0.0/24
--network_size=5000
--libvirt_type=kvm
```

Die einzelnen Optionen werden in den entsprechenden Abschnitten dieses Artikel aufgegriffen. Der Einfachheit halber konfigurieren Sie einmal eine *nova.conf* mit sämtlichen benötigten Einstellungen und verteilen diese dann auf alle entsprechenden Hosts.

Alternativ tragen Sie auf den einzelnen Hosts immer die dort zur Konfiguration nötigen Parameter ein und lassen den Rest aus. Eine Übersicht aller verfügbaren Parameter unter: <http://docs.openstack.org/cactus/openstack-compute/admin/content/reference-for-flags-in-nova-conf.html>

Zusätzlich zur oben genannten *nova.conf* existiert noch eine Datei mit dem Namen *api-paste.ini*, die die Beschreibung der API enthält. In dieser sollten üblicherweise keine Änderungen vorgenommen werden.

4.3.1 Datenbank

Zur Kommunikation mit der Datenbank wird das Python-SQL-Toolkit *SQLAlchemy* verwendet. Damit *Nova* auf die Datenbank zugreifen kann, passen Sie die entsprechende Zeile in der *nova.conf* an:

```
--sql_connection=mysql://root:testing@deimos/nova
```

In diesem Fall nutzen Sie eine MySQL-Datenbank namens *nova* auf dem Host *deimos*. Zur Verbindung verwenden Sie den Benutzer *root* mit dem Passwort *testing*. Von *Nova* können alle von *SQLAlchemy* unterstützten Datenbanken verwendet werden.

Hinweis: SQLAlchemy

Weiterführende Informationen zu *SQLAlchemy* finden sich unter <http://www.sqlalchemy.org>.

Von dem *API-Node* *ares* aus initialisieren Sie die Datenbank *nova* wie folgt:

```
ares:~ # nova-manage db sync
```

Hiermit werden die benötigten Tabellen in der Datenbank angelegt.

Zur Kontrolle, ob der Befehl wirklich erfolgreich war, lassen Sie sich direkt per MySQL die Struktur der Datenbank anzeigen, nachdem Sie sich mit ihr verbunden haben. Dies geschieht natürlich auf *deimos*:

```
deimos:~ # mysql -u root -p
Enter password:
[...]
mysql> use nova;
Database changed
mysql> show tables;
+-----+
```

```

| Tables_in_nova |
+-----+
| auth_tokens |
| certificates |
| console_pools |
| consoles |
| export_devices |
| fixed_ips |
| floating_ips |
| instance_actions |
| instances |
| iscsi_targets |
| key_pairs |
| migrate_version |
| networks |
| projects |
| quotas |
| security_group_instance_association |
| security_group_rules |
| security_groups |
| services |
| user_project_association |
| user_project_role_association |
| user_role_association |
| users |
| volumes |
+-----+
24 rows in set (0.00 sec)

```

Durch die rasante Entwicklung von *OpenStack* kann sich die obere Ausgabe unterscheiden.

Die in der Datenbank hinterlegten Informationen lassen sich auch dazu verwenden, eigene Auswertungsskripte zu schreiben.

Alternativ besteht die Möglichkeit, direkt mit `nova-manage` die Version der Datenbank zu prüfen:

```
# nova-manage db version
44
```

Die Datenbankversion für Cactus war 14, für Diablo 44.

4.3.2 AMQP / RabbitMQ

OpenStack verwendet das Python-Framework *carrot* zur Nutzung des AMQP Messaging. Unterstützt werden von diesem Framework nicht nur *RabbitMQ*, sondern auch *ZeroMQ* und *Apache ActiveMQ*. Von *OpenStack* wird derzeit ausschließlich *RabbitMQ* unterstützt, andere Systeme wurden bislang nicht getestet und werden nicht empfohlen.

Fügen Sie den Messaging Server zur Konfiguration hinzu:

```
--rabbit_host=deimos
```

Warnung: Neustart des *RabbitMQ* -Servers

Aktuell gibt es Probleme bei einem Neustart von *RabbitMQ* während *Nova* -Dienste darauf zugreifen. Teilweise können sich diese nicht wieder verbinden und müssen neu gestartet werden.

Aktuell wird vom *OpenStack* -Team ein anderes Verfahren zum Messaging entwickelt. Bevor dieser Teil aber produktiv genutzt werden kann, wird noch einige Zeit vergehen. Das entsprechende Paket heißt `openstack-burrow`.

4.3.3 Konfiguration der API

Zur Konfiguration der API-Node sind lediglich zwei Einträge in der `/etc/nova/nova.conf` nötig:

```
--ec2_url=http://ares:8773/services/Cloud  
--cc_host=ares
```

Diese werden für die Kommunikation mit der API genutzt.

4.3.4 Konfiguration des Objectstorages

Dieser Parameter weist *Nova* an, wo nach dem Objectstore gesucht werden soll.

```
--s3_host=ares
```

4.3.5 Virtualisierungsart

Über den Parameter `libvirt_type` geben Sie die gewünschte Virtualisierung an. Mögliche Werte sind:

- kvm
- xen
- qemu
- uml

Hier verwenden Sie KVM als Hypervisor. Also wird dieser Typ in `nova.conf` eingetragen:

```
--libvirt_type=kvm
```

Bisher bietet lediglich KVM Möglichkeiten zur Live-Migration von Maschinen.

4.3.6 Netzwerkkonfiguration

Zur Konfiguration des Netzwerkes für die Instanzen gibt es verschiedene Möglichkeiten:

- Flat Mode
- Flat DHCP Mode
- VLAN DHCP Mode

Zusätzlich unterstützt *Nova* das Konzept von Fixed IPs und Floating IPs. Fixed IPs werden vom Start einer Instanz bis zu ihrem Ende fest vergeben. Floating IPs hingegen können dynamisch einer Instanz zugeordnet werden.

Bei der Nutzung von *Flat Mode* definiert man ein Subnetz und die entsprechende IP-Adresse wird beim Start der Instanz in diese injiziert. Hierbei muss die vollständige Netzkonfiguration (Bridges etc.) vom Administrator manuell vorgenommen werden.

Der *Flat DHCP Mode* hingegen muss man die Netzwerkbrücke auch manuell konfigurieren. Allerdings startet *Nova* dann einen DHCP-Server auf dem entsprechenden Interface, um so die Netzkonfiguration dem Gast zur Verfügung zu stellen.

Im Default-Modus *VLAN* erzeugt `compute` dynamisch Netzwerkbrücken und VLANs pro Projekt. Dabei muss die verwendete Hardware (Switches) natürlich das entsprechende VLAN-Tagging beherrschen. Für jedes VLAN wird ein eigener DHCP-Server gestartet, der die Konfiguration verteilt.

Parameter in `/etc/nova/nova.conf`:

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--dhcp_domain=example.com
--network_manager=nova.network.manager.FlatDHCPManager
--flat_network_dhcp_start=192.168.3.2
--flat_injected=False
```

Der Parameter `flat_injected` schaltet die IPv6-Konfiguration für die Hosts aus.

4.3.7 Starten der Dienste

Die einzelnen Komponenten von *Nova* können direkt an den einzelnen Hosts bequem mit entsprechenden Start-Stop-Skripten verwaltet werden. Diese Skripte befinden sich wie unter SLES üblich in `/etc/init.d`:

Datei	Aufgabe
<code>openstack-nova-api</code>	startet / stoppt nova-api
<code>openstack-nova-compute</code>	startet / stoppt nova-compute
<code>openstack-nova-network</code>	startet / stoppt nova-network
<code>openstack-nova-scheduler</code>	startet / stoppt nova-scheduler
<code>openstack-nova-volume</code>	startet / stoppt nova-volume
<code>openstack-nova-objectstore</code>	startet / stoppt nova-objectstore

Natürlich sind die entsprechenden Dateien lediglich auf den passend konfigurierten Hosts verfügbar.

```
# /etc/init.d/openstack-nova-api start
Starting OpenStack::Nova api server           done
```

Für den "schnellen" Start wird der folgende Befehl genutzt:

```
# for i in {api,compute,network,scheduler}; do /etc/init.d/openstack-
nova-$i start; done
```

Die pro Host benötigten Dienste müssen wieder per `chkconfig` aktiviert werden, damit die Dienste beim Systemstart gestartet werden. Der Dienst für `nova-volume` wird an dieser Stelle absichtlich ausgelassen, da die benötigte Konfiguration später erfolgt.

4.3.8 Logfiles

Die Logfiles der einzelnen Dienste liegen unter `/var/log/nova`; für jeden Dienst gibt es eine eigene Datei:

Datei	Dienst
api.log	nova-api
compute.log	nova-compute
network.log	nova-network
volume.log	nova-volume
objectstore.log	nova-objectstore
scheduler.log	nova-scheduler

Um festzustellen, ob die gerade gestarteten Dienste ordnungsgemäß funktionieren, werfen Sie einen Blick in die entsprechenden Dateien. Eventuell auftretende Fehlermeldungen können im Internet gesucht werden, vielfach existieren schon passende Antworten im Launchpad (<https://launchpad.net/openstack>), wo auch der Sourcecode von *OpenStack* verwaltet wird.

Hinweis: AMQP server

Sollte der Fehler `ERROR nova.rpc [-] AMQP server on localhost:5672 is unreachable. Trying again in 10 seconds.` in der Datei `/var/log/messages` auftauchen, überprüfen Sie die Konfiguration des Queueings. Voraussichtlich fehlt in `/etc/nova/nova.conf` der Eintrag `-rabbit_host`.

4.4 Kommandübersicht

Die Kommunikation mit *Nova* findet über die API statt. Mit Hilfe der entsprechenden Werkzeuge aus den unterschiedlichen Paketen ist es relativ einfach *Nova* zu verwalten. Die folgende Tabelle enthält eine Übersicht der in diesem Artikel verwendeten Werkzeuge, ihrer Funktion und aus welchem Paket sie stammen.

Kommando	Paket	Beschreibung
euca-add-keypair	euca2ools ¹	einen ssh-key der Datenbank hinzufügen
euca-describe-keypairs	euca2ools	eingetragene Schlüssel anzeigen
euca-authorize	euca2ools	Aktionen erlauben, Einstellungen speichern
euca-describe-groups	euca2ools	Anzeigen von Gruppen
euca-describe-images	euca2ools	Verfügbare Images anzeigen
euca-describe-instances	euca2ools	Aktuell verwaltete Instanzen zeigen
euca-run-instances	euca2ools	Instanzen starten
euca-bundle-image	euca2ools	Imagedateien bündeln
euca-upload-bundle	euca2ools	Bündel in den Store laden
euca-register	euca2ools	Imagedateien registrieren
glance-upload	<i>Glance</i>	Imagedateien zu <i>Glance</i> hinzufügen
nova-manage	<i>Nova</i>	<i>Nova</i> -Verwaltungswerkzeug (Benutzer, Projekte etc.)
nova	<i>Nova</i>	verwalten von <i>Nova</i> , Instanzen etc.

4.5 nova-manage

`nova-manage` dient zur Verwaltung von Ressourcen in *Nova* . Darüber werden u.a. Projekte, Benutzer, Netzwerke etc. angelegt, verändert oder entfernt.

Auch die Datenbank wird erstmals mit `nova-manage` initialisiert und bei eventuellen Updates, die Änderungen an der Datenbank vornehmen, muss dieses erneut mit dem Parameter `db sync` aufgerufen werden. Dieser Schritt wurde im oberen Teil beim Anlegen der Datenbank bereits erledigt (vgl. Datenbank, Seite 20).

4.5.1 Benutzer

Zur einfachen Verwaltung der Instanzen erzeugen Sie einen `admin` mit einem passenden Benutzernamen:

```
ares:~ # nova-manage user admin berendt
2011-02-10 12:41:27,677 AUDIT nova.auth.manager [-] Created user
  berendt (admin: True)
export EC2_ACCESS_KEY=f0780055-ddb1-4216-9235-ad920f585603
export EC2_SECRET_KEY=e50ace36-c617-4801-8cc7-0f7b890c6118
```

Obiges Kommando erzeugt den Benutzer `berendt`, dabei wird der Eintrag in die Datenbank geschrieben und die benutzerspezifischen Access-Keys erzeugt. Diese werden später benötigt, können aber jederzeit abgerufen werden.

¹ein Großteil der `euca2ools` setzen exportierte EC2-Variablen voraus, um zu funktionieren (vgl. 4.5.4, Zertifikate exportieren)

4.5.2 Projekte

Instanzen werden in Projekten verwaltet. Pro Projekt muss ein Administrator existieren, der das Projekt verwalten kann. Mit folgenden Parametern von *nova-manage* wird das Projekt *testing* erzeugt und der im vorherigen Abschnitt erzeugte Benutzer als Manager eingetragen.

```
ares:~ # nova-manage project create testing berendt
```

4.5.3 Netzwerke

Für das konfigurierte Netzwerkverfahren benötigen Sie einen Pool an IP-Adressen, die den Instanzen zugewiesen werden können.

Dieser Pool muss in der Datenbank eingetragen werden, ansonsten weiß *Nova* nicht, welche Adressen verwendet werden können. Außerdem wird bei den entsprechenden Datensätzen vermerkt, welche Adresse bereits in der Nutzung ist. Um eventuell bereits belegte IP-Adressen später auszuschließen, setzen Sie direkt in der Datenbank bei der entsprechenden Adresse die Spalte *reserved* auf 1, dann wird diese Adresse nicht per DHCP verteilt.

```
ares:~ # nova-manage network create public 192.168.3.0/24 1 256
```

Sollte an dieser Stelle ein Fehler auftauchen und das Netzwerk nicht erzeugt werden, so fehlt aktuell der Parameter *-bridge_interface*. Diesen kann man einfach an den Befehl anhängen:

```
# nova-manage network create public 192.168.0.0/24 1 256 --
  bridge_interface=br0
```

Hierbei nimmt das Programm dem Benutzer die Arbeit ab. Mit diesem Aufruf werden passend für das gewünschte Class-C-Netz die einzelnen IP-Adressen als Datensätze in der Datenbank angelegt, mit der übergebenen Maske.

Die erzeugten Netzwerke können per *nova-manage network list* angezeigt werden:

```
# nova-manage network list
id      IPv4                IPv6                start address      DNS1
          DNS2                VlanID              project             uuid
1       192.168.0.0/24      None                192.168.0.3        None
          None                100                 None                None
```

4.5.4 Zertifikate exportieren

Zur Nutzung der *euca-tools* benötigen Sie die entsprechenden Umgebungsvariablen. Diese werden mit Hilfe von *nova-manage* exportiert und in eine Datei *novarc* geschrieben. Zu Beginn einer Sitzung lassen Sie diese einlesen, damit die Umgebungsvariablen gesetzt und genutzt werden können:

```
ares:~ # mkdir -p /root/creds

ares:~ # nova-manage project zipfile testing berendt /root/creds/
novacreds.zip
2011-02-10 12:43:06,419 WARNING nova.auth.manager [-] No vpn data for
project testing

ares:~ # unzip /root/creds/novacreds.zip -d /root/creds/
Archive: /root/creds/novacreds.zip
  extracting: /root/creds/novarc
  extracting: /root/creds/pk.pem
  extracting: /root/creds/cert.pem
  extracting: /root/creds/cacert.pem

ares:~ # source /root/creds/novarc
```

Für die häufige Nutzung bietet es sich an, das Einlesen der `novarc` in einer entsprechenden Datei der Shell einzutragen.

4.5.5 SSH Key

Mit Hilfe der `euca-tools` erzeugen Sie einen SSH Key. Dabei wird der Schlüssel in die Datenbank eingetragen und als Datei abgelegt. Auf diese Datei setzen Sie eingeschränkte Leserechte, sonst gibt der SSH-Zugriff später eine Fehlermeldung aus. Wichtig ist, vor dem Nutzen der `euca-tools` die Umgebungsvariablen wie im vorhergehenden Abschnitt gezeigt, eingelesen zu haben.

```
ares:~ # euca-add-keypair testing > testing.priv
ares:~ # chmod 600 testing.priv
```

Um zu kontrollieren, dass der Schlüssel korrekt eingetragen wurde, wird `euca-describe-keypairs` genutzt:

```
# euca-describe-keypairs
KEYPAIR testing a6:2a:92:0c:13:20:a0:f7:ff:97:da:0a:29:27:39:fb
```

Wichtig: Proxyserver

Bei Verwendung sämtlicher `euca-tools` darf keine Umgebungsvariable für einen Proxyserver gesetzt sein (`export http_proxy`)! Andernfalls brechen die Programme mit einem unbekanntem Fehler ab. Zusätzlich darf beim Starten der einzelnen *Nova* -Dienste auch kein Proxyserver gesetzt sein!

4.5.6 Security Groups

Zur späteren Erreichbarkeit der Instanzen schalten Sie ICMP und SSH-Zugriff frei. Erzeugen Sie die passenden Einträge.

1. SSH-Zugriff auf die virtuelle Maschine erlauben:

```
ares:~ # euca-authorize -P tcp -p 22 default
GROUP    default
PERMISSION    default  ALLOWS  tcp    22    22
```

2. ICMP ping erlauben:

```
ares:~ # euca-authorize -P icmp -t -1:-1 default
GROUP    default
PERMISSION    default  ALLOWS  icmp   -1    -1
```

3. Einstellungen kontrollieren:

```
ares:~ # euca-describe-groups
GROUP    testproject    default default
PERMISSION    testproject    default  ALLOWS  tcp    22    22
           FROM    CIDR    0.0.0.0/0
PERMISSION    testproject    default  ALLOWS  icmp   -1    -1
           FROM    CIDR    0.0.0.0/0
```

4.5.7 service list

Der *Object Storage* ist für die Ablage von Images notwendig, aus denen später die virtuellen Systeme erstellt werden.

Konfigurationserweiterung `/etc/nova/nova.conf`:

```
--s3_host=ares
```

Hinweis: iptables

Sollte die Firewall nicht deaktiviert worden sein, muss man auf die Regeln des Paketfilters achten. Automatisch erstellte iptables-Regeln blocken den TCP-Port 3333, der für den Dienst `nova-objectstore` benötigt wird. Daher müssen Sie diesen nach dem Start manuell mit

```
iptables -I INPUT -p tcp -dport 3333 -j ACCEPT
```

freigeben.

Für erste Tests ist es ausreichend, den in *Nova* vorhandenen Dienst `nova-objectstore` zu verwenden. Dieser legt Images lokal unter `/var/lib/nova/images` ab und ist über ein S3-API ansprechbar. In der Produktion sollte er später durch *Glance* und *Swift*, welche nachfolgend vorgestellt werden, ersetzt werden.

4.6 Anbindung von glance

Im Kapitel 3 wurde die Konfiguration und Nutzung von *Glance* dargestellt. Hier wird *Glance* nun an *Nova* angebunden, damit die Images von *Glance* verwaltet werden.

Damit *Nova* die Images von *Glance* bezieht muss die Konfiguration von *Nova* angepasst werden. Dazu werden folgende Optionen in der `/etc/nova/nova.conf` gesetzt:

```
--image_service=nova.image.glance.GlanceImageService
--glance_api_servers=chronos:9292
```

Nun werden die *Compute* -Nodes ihre Images von *Glance* beziehen.

4.7 Verfügbare Images anzeigen

4.7.1 euca2ools

Zur Kontrolle, welche Images und Typen aktuell für *Nova* zur Verfügung stehen, greifen Sie erneut auf die *euca2ools* zurück. Das Werkzeug *euca-describe-images* liefert einen Überblick. Zur Verwendung müssen die entsprechenden Umgebungsvariablen gesetzt sein, die bereits im Abschnitt *Zertifikate exportieren* (vgl. 4.5.4) erzeugt wurden.

```
ares:~ # euca-describe-images
IMAGE  ami-00000003      None (ttylinux)          available      public
              machine ari-00000002      aki-00000001
IMAGE  ami-00000002      None (ttylinux)          available      public
              machine
IMAGE  ami-00000001      None (ttylinux)          available      public
              machine
```

4.7.2 nova

Alternativ besteht die Möglichkeit, die existierenden Images per *nova* abzufragen:

```
# nova image-list
+-----+-----+-----+
| ID | Name | Status |
+-----+-----+-----+
| 1 | ttylinux | ACTIVE |
| 2 | ttylinux | ACTIVE |
| 3 | ttylinux | ACTIVE |
+-----+-----+-----+
```

Hier können zusätzliche Details zum Image abgefragt werden, dafür wird lediglich die ID benötigt, die in der vorhergehenden Ausgabe ersichtlich ist:

```
# nova image-show 3
+-----+-----+-----+
| Property | Value |
+-----+-----+-----+
| created | 2011-09-12T13:42:12Z |
| id | 3 |
| | |
```



```

chronos:~ # euca-run-instances -t ml.tiny -k testing -n 2 ami-00000003
RESERVATION      r-msramq0g      testproject      default
INSTANCE         i-00000001      ami-00000003      pending
  testing (testproject, brontes)  2011-09-14T10:57:47Z  aki
-00000001      ari-00000002
INSTANCE         i-00000002      ami-00000003      pending
  testing (testproject, None)      2011-09-14T10:57:47Z  aki
-00000001      ari-00000002

```

Es stehen unterschiedliche Typen für die Größe der virtuellen Maschinen zur Verfügung. Diese werden über den Parameter `-t` definiert. Der Parameter `-n` weist `euca-run-instances` an, eine entsprechende Anzahl von Instanzen zu starten, hier sind es zwei. Der Parameter `-k` übergibt den zu nutzenden `ssh`-Schlüssel der Instanz. Vor dem Start wird dieser in das kopierte Image geschrieben, so ist in diesem Beispiel ein Login in die virtuelle Maschine mit dem Befehl `ssh -i testing.priv root@<IP>` möglich. Die IP-Adresse erhält man durch einen erneuten Aufruf von `euca-describe-instances`, wenn die virtuelle Maschine gestartet wurde.

Nach einigen Sekunden können Sie sich den Status der Instanzen mit `euca-describe-instances` anzeigen lassen. Dabei erfahren Sie den Typ, die zugewiesene IP-Adresse und in der zweiten Spalte auch die eindeutige Instanz-ID. Diese wird auch als Bezeichnung vom verwendeten Hypervisor genutzt. So können Sie sich an den *Compute* -Nodes z.B. mit `virsh list` die Instanzen anzeigen lassen und werden wieder auf die ID stoßen.

Im folgenden ein Beispiel für eine stärkere Auslastung:

```

chronos:~ # euca-describe-instances
RESERVATION      r-okdc1w0u      testing default
INSTANCE         i-00000e8f      ami-j7xy0bqw      192.168.3.29
  192.168.3.29 launching      None (testing, ares)  0      ml.tiny
  2011-03-17T09:16:13Z      nova
INSTANCE         i-00000e90      ami-j7xy0bqw      192.168.3.30
  192.168.3.30 running      None (testing, brontes)  1
  ml.tiny 2011-03-17T09:16:13Z      nova
INSTANCE         i-00000e91      ami-j7xy0bqw      192.168.3.32
  192.168.3.32 launching      None (testing, ares)  2      ml.tiny
  2011-03-17T09:16:13Z      nova
INSTANCE         i-00000e92      ami-j7xy0bqw      192.168.3.31
  192.168.3.31 launching      None (testing, ares)  3      ml.tiny
  2011-03-17T09:16:13Z      nova
RESERVATION      r-mne5so6o      testing default
INSTANCE         i-00000e8e      ami-j7xy0bqw      192.168.3.25
  192.168.3.25 running      testing (testing, brontes)  0
  ml.tiny 2011-03-16T15:16:01Z      nova

```

4.8.2 Starten per nova

Mit Hilfe des Kommandos `nova` können Instanzen direkt über die Nova-API gestartet werden:

```
# nova boot --image 3 --flavor 1 NeuerTest
+-----+-----+-----+-----+-----+-----+-----+
| Property | Value |
+-----+-----+-----+-----+-----+-----+-----+
| accessIPv4 | |
| accessIPv6 | |
| adminPass | sFztDYbKNadiqAJf |
| config_drive | |
| created | 2011-09-13T09:02:14Z |
| flavor | m1.tiny |
| hostId | |
| id | 4 |
| image | ttylinux |
| key_name | None |
| metadata | {} |
| name | NeuerTest |
| progress | 0 |
| status | BUILD |
| updated | 2011-09-13T09:02:15Z |
| uuid | a4d57b0e-d064-4989-a66b-a1ca94571839 |
+-----+-----+-----+-----+-----+-----+-----+
```

Die ID des Parameters `image` ist die des zu startenden Images. Es existiert ein Parameter `--key`, dieser scheint aktuell den ssh-Schlüssel leider nicht in das Image zu schreiben. Für das Injizieren des Schlüssels muss man also vorerst noch auf die `euca-tools` zurückgreifen.

Die Liste mit den Typen, hier genannt *flavor* wird mit dem Befehl `nova flavor-list` ausgegeben:

```
# nova flavor-list
+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | Memory_MB | Swap | Local_GB | VCPUs | RXTX_Quota |
| RXTX_Cap |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | m1.tiny | 512 | | 0 | | |
| 2 | m1.small | 2048 | | 20 | | |
| 3 | m1.medium | 4096 | | 40 | | |
| 4 | m1.large | 8192 | | 80 | | |
| 5 | m1.xlarge | 16384 | | 160 | | |
+-----+-----+-----+-----+-----+-----+-----+
```

4.8.3 Stoppen per euca2ools

Um mit den `euca2ools` Instanzen zu stoppen existiert das Kommando `euca-terminate-instances`. Diesem wird die Instanz-ID übergeben:

```
# euca-terminate-instances i-0000000c
```

Dadurch wird die Instanz beendet.

4.8.4 Stoppen per nova

Es besteht ausserdem die Möglichkeit, per `nova` Instanzen zu beenden. Im Folgenden wird erst eine Ausgabe der laufenden Instanzen erzeugt um dann die Instanz mit der ID 11 zu löschen:

```
# nova list
+-----+-----+-----+-----+
| ID | Name | Status | Networks |
+-----+-----+-----+-----+
| 11 | Server 11 | ACTIVE | public=192.168.0.3 |
+-----+-----+-----+-----+
# nova delete 11
```

4.9 Volumes

Es besteht die Möglichkeit, Daten einer Instanz zu erhalten. Dazu müssen diese auf sogenannte *persistente Volumes* gespeichert werden. Diese Volumes werden nach dem Start einer virtuellen Maschine dieser zugewiesen. Es ist nicht möglich, aus mehreren Instanzen zeitgleich auf ein solches Volume zuzugreifen. Derartige Setups realisieren Sie alternativ mit *NFS*.

4.9.1 Vorbereitung

Für die Nutzung der Komponente `nova-volume` benötigen Sie die `iscsitarget`-Pakete. Installieren Sie diese mit folgendem Aufruf:

```
# yast2 iscsi-server
```

YaST2 löst alle Abhängigkeiten auf und installiert diese auch. Den Assistenten beenden Sie mit *Finish*.

Aktivieren Sie das *iscsitarget*:

```
# rciscsitarget start
# chkconfig iscsitarget on
```

Zusätzlich benötigt `nova-volume` noch eine LVM Volume Group mit dem Namen `nova-volumes`. Diese muss natürlich konfiguriert sein. Im Folgenden ein Beispiel zum Anlegen einer passenden Partition und der zugehörigen Volume Group.

Bereiten Sie eine Partition auf einer Festplatte vor. Hier wird die erste Partition auf `/dev/sdb` erzeugt und der Typ auf Linux LVM gesetzt:

```
# fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or
  OSF disklabel
Building a new DOS disklabel with disk identifier 0xee2e5847.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

The number of cylinders for this disk is set to 8000.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
 1) software that runs at boot time (e.g., old versions of LILO)
 2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)
Warning: invalid flag 0x0000 of partition table 4 will be corrected by
  w(rite)

Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-8000, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-8000, default 8000):
Using default value 8000

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 8e
Changed system type of partition 1 to 8e (Linux LVM)

Command (m for help): p

Disk /dev/sdb: 8388 MB, 8388608000 bytes
64 heads, 32 sectors/track, 8000 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes
Disk identifier: 0xee2e5847

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            1         8000     8191984    8e  Linux LVM

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
```

Syncing disks.

Erzeugen Sie ein für LVM nutzbares physisches Volume:

```
# pvcreate /dev/sdb1
No physical volume label read from /dev/sdb1
Physical volume "/dev/sdb1" successfully created
```

Erstellen Sie die benötigte Volume Group nova-volumes:

```
# vgcreate nova-volumes /dev/sdb1
Volume group "nova-volumes" successfully created
```

Damit sind die Vorbereitungen für die Nutzung von nova-volume abgeschlossen.

4.9.2 Erzeugen und Aktivieren

Damit ein Volume einer Instanz zur Verfügung gestellt werden kann, muss es erst erzeugt werden. Dabei werden Größe und Zone angegeben. Da *Nova* aktuell nur eine Zone hat, ist dies nova.

Hier wird ein Volume mit der Größe von einem Gigabyte erzeugt:

```
# euca-create-volume -s 1 -z nova
VOLUME vol-00000003 1 creating (testproject, None, None, None
) 2011-09-15T06:47:15Z
```

Zur Kontrolle des Status wird *euca-describe-volumes* genutzt:

```
# euca-describe-volumes
VOLUME vol-00000003 1 nova available (testproject,
None, None, None) 2011-09-15T06:47:15Z
```

Mögliche Statusmeldungen für Volumes sind:

creating das Volume wird aktuell erstellt

error beim Erstellen des Volumes trat ein Fehler auf

available das Volume wurde erfolgreich erstellt und ist nutzbar

Sobald der Status auf „available“ steht, ist das Volume nutzbar und kann in eine laufende Instanz eingehängt werden:

```
# euca-attach-volume -i i-0000000c -d /dev/vdb vol-00000003
VOLUME vol-00000003
```

Der Parameter *-i* beschreibt dabei die Bezeichnung der Instanz, diese wird über *euca-describe-instances* ausgelesen. *-d* gibt das Device vor, als das das Volume der Instanz zugewiesen werden soll. Der letzte Parameter schliesslich ist die Bezeichnung des Volumes.

Ein erneuter Aufruf von *euca-describe-volumes* gibt nun auch den *Compute* -Host an, an den das Volume zugewiesen wurde:

```
# euca-describe-volumes
VOLUME vol-00000003 1 nova available (testproject,
    Brontes, None, None) 2011-09-15T06:47:15Z
```

4.9.3 Entfernen

Volumes können per `euca-detach-volume` aus einer Instanz entfernt werden:

```
# euca-detach-volume vol-00000003
VOLUME vol-00000003
```

Damit steht das Volume der Instanz nicht mehr zur Verfügung und könnte nun einer anderen Instanz zugewiesen werden.

4.9.4 Löschen

Achtung: Datenverlust beim Löschen von Volumes

Das Löschen von Volumes führt zu Datenverlust. Sollten Sie Daten auf einem zu entfernenden Volume noch benötigen, sichern Sie diese gegebenenfalls an eine andere Stelle *bevor* Sie das Volume endgültig entfernen.

Zur Darstellung aller Volumes wird `euca-describe-volumes` verwendet:

```
# euca-describe-volumes
VOLUME vol-00000003 1 nova available (testproject,
    None, None, None) 2011-09-15T06:47:15Z
```

Um ein nicht mehr genutztes und einer Instanz zugewiesenes Volume zu entfernen, wird `euca-delete-volume` genutzt:

```
# euca-delete-volume vol-00000003
VOLUME vol-00000003
```

Nach kurzer Zeit, in der das logische Volume gelöscht wird, erscheint dieses nun nicht mehr in der Ausgabe von `euca-describe-volumes`, in diesem Beispiel eine leere Ausgabe des Aufrufs:

```
# euca-describe-volumes
#
```

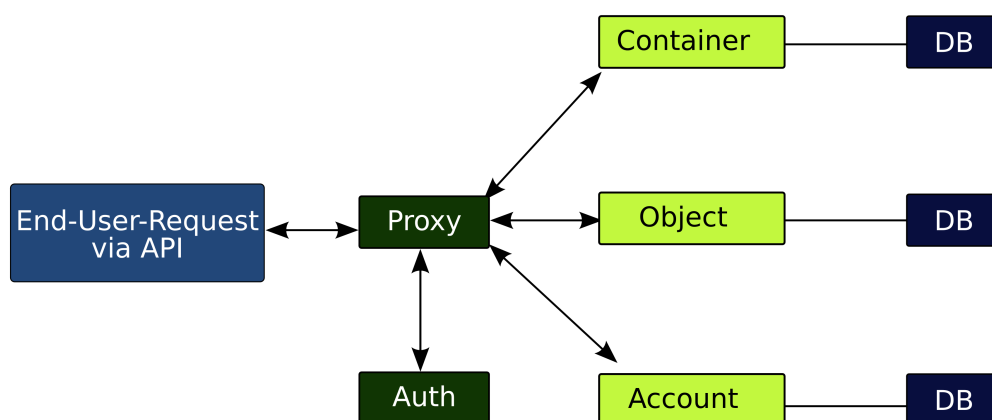
5 OpenStack Object Storage - swift

5.1 Übersicht

Bei OpenStack Object Storage (swift) handelt es sich um einen Objectstore, ähnlich Amazons S3. Hier können über eine API Daten hinterlegt werden. Es besteht keine Möglichkeit, diesen Objectstore als direkten Dateiserver zu benutzen, also z.B. wie das Network File System (NFS).

Sollte sich der verfügbare Speicherplatz des Objectstores dem Ende zuneigen, besteht die Möglichkeit, einen weiteren Rechner hinzuzufügen und dadurch den verfügbaren Speicherplatz zu mehren. Aus diesem Grund spricht man von unendlichem Speicher (infinite storage).

Zur Erhaltung der Redundanz und aus Gründen der Skalierbarkeit werden die Daten mehrfach auf Speichersystemen geschrieben. Dabei wird eine Einteilung in sogenannte Zonen vorgenommen. Diese können logisch oder physikalisch getrennt sein. Auch gibt es keinen zentralen Ort für die benötigte Benutzerverwaltung, jeder dem Objectstore zugehörige Rechner führt die Accountdatenbank mit sich.



Jede Komponente speichert ihre Daten lokal (auf jedem beteiligten Host) in einer Datenbank.

5.1.1 Proxy Server

Der Proxyserver hält die anderen Swift-Komponenten zusammen. Für jede Anfrage wird er den Account, den Container oder das Objekt im Ring lokalisieren und entsprechend zuweisen. Auch die API arbeitet durch den Proxyserver.

5.1.2 Ringe

Ein Ring repräsentiert die Zuordnung zwischen der physikalischen Position und dem gespeicherten Namen. Es existieren separate Ringe für Accounts, Container und Objekte. Sollten Komponenten Operationen an Objekten, Container oder Accounts durchführen, müssen sie mit dem entsprechenden Ring interagieren, um die Position des Zielobjektes im Ring herauszufinden.

Mit Hilfe von Zonen, Geräten und Partitionen sowie Replikationen verwaltet ein Ring seine Zuordnungen. Jede Partition im Ring wird dreimal über den Cluster repliziert, die Position der Partition wird in der Zuordnung des Rings gespeichert. Zusätzlich ist der Ring für die Zuordnung von Geräten bei einem Ausfall zuständig.

Zonen separieren die Daten im Ring. Jede Replika einer Partition ist garantiert in einer anderen Zone gespeichert. Eine Zone kann dabei ein Laufwerk, ein Server oder gar ein gesamtes Datenzentrum sein.

5.1.3 Object Server

Beim *Object Server* handelt es sich um einen einfachen *Blob Storage*, der Objekte auf lokalen Geräten empfangen, speichern und löschen kann. Dabei werden die Objekte als Binärdateien auf dem Dateisystem mit zusätzlichen Metadaten gespeichert, diese wiederum werden in den erweiterten Attributen (`xattr`) gehalten. Dieses Verfahren erfordert natürlich ein darunterliegendes Dateisystem, das `xattr` unterstützt.

Hinweis: Dateisysteme

Bei einigen Dateisystemen, u.a. `ext3`, ist die Unterstützung von `xattr` standardmäßig abgeschaltet. Diese Attribute müssen beim Mounten des Dateisystems aktiviert werden.

Jedes Objekt wird in einem Pfad, der sich aus dem Hash des Objektnamens und einem Zeitstempel der Dateioption zusammengesetzt, gespeichert. Dabei gewinnt immer der *last write*. Dieser stellt zusätzlich sicher, dass die letzte Version eines Objektes zur Verfügung steht. Ein Löschvorgang wird ebenso als eine Dateiversion gehandhabt (eine 0 Byte große Datei mit der Endung `.ts` (engl. *tombstone*)). Dies sorgt dafür, dass auch gelöschte Dateien immer korrekt repliziert werden. So tauchen im Fehlerfall auch nicht auf magische Art und Weise ältere Dateiversionen auf.

5.1.4 Container Server

Der *Container Server* ist dafür zuständig, Objektlisten zu verwalten. Er weiß selbst nicht, wo die Objekte sind, lediglich welcher Container welche Objekte enthält. Dabei speichert er diese Listen als einzelne `sqlite`-Datenbankdateien ab und repliziert diese auf den Cluster, ähnlich wie es mit den Objekten selbst gemacht wird.

5.1.5 Account Server

Der *Account Server* ist dem *Container Server* recht ähnlich. Im Gegensatz zu diesem ist er für die Listings von Containern zuständig, nicht dem von Objekten.

5.1.6 Replikation

Per Replikation wird das System in einem konsistenten Zustand gehalten, auch wenn es vorübergehende Fehler wie Netzwerkprobleme oder Gerätefehler gibt.

Der Replikationsprozess vergleicht die lokalen Daten mit jeder entfernten Kopie und trägt Sorge, dass alle die letzte Version enthalten. Objektreplikation nutzt Hash-Listen, um schnelle Vergleiche einzelner Teile von Partitionen zu machen. Die Replikation von Containern und Accounts hingegen nutzt eine Kombinationen aus Hashes und Wasserzeichen.

Die gesamte Replikation arbeitet im *Push*-Verfahren. Die Replikation von Objekten ist lediglich ein `rsync` von Dateien. Fehlende Einträge werden per `http` von Account und Container auf die anderen Hosts verteilt, ganze Datenbankdateien per `rsync`.

Zusätzlich werden mit der Replikation Löschvorgänge sichergestellt. Wird ein Objekt (Objekt, Container oder Account) gelöscht, wird der oben erwähnte *tombstone* erzeugt und als letzte Version des Objektes genutzt. Der Replikator sieht diesen Eintrag und entfernt das zu löschende Objekt vom gesamten System.

5.1.7 Updates

Manchmal können Container- oder Account-Daten nicht sofort gespeichert und verteilt werden. Dies tritt normalerweise im Fehlerszenario oder bei sehr hoher Auslastung auf. Sollte ein Update fehlschlagen, wird es auf dem lokalen Dateisystem in eine Warteschlange geschoben. Dann wird der *Updater* die fehlerhaften Prozesse abarbeiten. Stellen wir uns vor, ein Container Server ist unter hoher Last und ein neues Objekt wird in das System geschoben. Für Lesezugriffe steht das Objekt sofort zur Verfügung, allerdings hat der Container Server sein Objektverzeichnis nicht aktualisiert. Dieser Prozess wandert in die Warteschlange und wird dann vom Updater ausgeführt. Container-Verzeichnisse können unter Umständen nicht sofort das Objekt enthalten.

In der Praxis ist dieses Zeitfenster nur so groß wie der Abstand, in dem der Updater läuft und wird wahrscheinlich vom Proxyserver nicht bemerkt.

5.1.8 Auditors

Auditors gehen den lokalen Server durch und überprüfen dabei die Integrität von Objekten, Containern und Accounts. Falls ein Defekt gefunden wird, wird das Objekt in Quarantäne verschoben und die Replikation ersetzt die Datei von einer Replika. Andere Fehler werden protokolliert.

5.2 Vorbereitung

Diese Schritte sind auf den beteiligten Systemen notwendig. Sie konfigurieren Swift nur einmal und kopieren die erzeugten Dateien später auf die beteiligten Systeme.

5.2.1 Partitionierung

Zum einfach Test von *Swift* werden lediglich zwei Partition auf zwei Nodes als Storage angebunden.

Dies muss im produktiven Umfeld selbstverständlich anders gehandhabt werden, insbesondere ist dort die Verwendung von mindestens fünf Zonen laut den Entwicklern empfohlen.

Auf der Festplatte `/dev/sda` richten Sie eine weitere Partition ein, Typ `Linux`. Diese formatieren Sie – wie von *OpenStack* empfohlen – mit dem XFS-Dateisystem. Dieser Schritt erfolgt auf beiden beteiligten Hosts:

```
# mkfs.xfs -i size=1024 /dev/sda5
Metadaten =/dev/sda5          isize=1024    agcount=4, agsize
           =3529780 blks
           =                      sectsz=512    attr=2
Daten      =                      bsize=4096   Blöcke=14119119, imaxpct
           =25
           =                      sunit=0      swidth=0 blks
Benennung =Version 2          bsize=4096   ascii-ci=0
Protokoll  =Internes Protokoll bsize=4096   Blöcke=6894, Version=2
           =                      sectsz=512    sunit=0 blks, lazy-count
           =1
Echtzeit   =keine             extsz=4096   Blöcke=0, rtextents=0
```

Tragen Sie die neuangelegte Partition in `/etc/fstab` ein, damit sie beim Systemstart automatisch eingehängt wird:

```
# echo -e "/dev/sda5\t /srv/node/sda5\t xfs\t noatime,nodiratime,
           nobarrier,logbufs=8\t 0 0" >> /etc/fstab
```

Erstellen Sie den Mountpunkt `/srv/node/sda5`:

```
# mkdir -p /srv/node/sda5
```

Mounten Sie das Dateisystem, um zu kontrollieren, ob alles funktioniert:

```
# mount
[...]
/dev/sda5 on /srv/node/sda5 type xfs (rw,noatime,nodiratime,nobarrier,
logbufs=8)
```

Damit sind die notwendigen Vorbereitungen für das Dateisystem abgeschlossen.

5.2.2 rsync

```
/etc/rsyncd.conf
```

Passen Sie auf allen beteiligten Hosts die Datei `/etc/rsyncd.conf` an. *Swift* kommuniziert auch über `rsync`. Das folgende Beispiel zeigt die vollständige Konfiguration. Die IP-Adresse muss selbstverständlich angepasst werden:

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 192.168.0.1

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

Nach dem Ändern starten Sie den Daemon neu und konfigurieren Sie dessen Aktivierung bei Systemstart:

```
# rcrsyncd restart
# chkconfig rsyncd on
```

Kontrollieren Sie die vorgenommenen Einstellungen:

```
# rsync rsync://172.19.134.24
```

Hier sollte eine solche Ausgabe erscheinen:

```
account
container
object
```

Der erfolgreiche Test schliesst die Vorbereitung von `rsyncd` ab.

5.3 Installation

Installieren Sie auf dem ersten Host alle Komponenten, damit er als Storage-Node funktioniert. Zusätzlich installieren Sie die Komponente `swift-proxy`, über die die Kommunikation mit der API stattfindet. In einem produktiven Setup besteht die Möglichkeit, mehrere Proxy Server aufzusetzen.

```
# zypper in openstack-swift openstack-swift-account openstack-swift-
  container openstack-swift-object openstack-swift-proxy
```

Für den anderen beteiligten Host installieren Sie folgende Pakete:

```
# zypper in openstack-swift openstack-swift-account openstack-swift-
  container openstack-swift-object
```

Die Abhängigkeiten werden `memcached` und eine neuere Variante von `sqlite3` installieren, da diese zwingend benötigt werden.

5.4 Konfiguration

Die Konfiguration von Swift verteilt sich über die bereits genannten Bestandteile. Eine Konfiguration für den `account-server`, eine für den `container-server` und eine Konfiguration für den `object-server`. Zusätzlich wird eine `proxy-server.conf` und eine generelle `swift.conf` benötigt.

Legen Sie die Datei `/etc/swift/swift.conf` an und füllen Sie diese mit einem Hash:

```
[swift-hash]
swift_hash_path_suffix = changeme
```

Der dargestellte Hash sollte natürlich geändert werden.

5.4.1 Proxy Server

Der Proxy Server stellt die API zur Verfügung, die Konfiguration selbst findet in der Datei `/etc/swift/proxy-server.conf` statt:

```
[DEFAULT]
# Enter these next two values if using SSL certifications
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
bind_port = 8080
workers = 8
user = swift

[pipeline:main]
# keep swauth in the line below if you plan to use swauth for
  authentication
#pipeline = healthcheck cache swauth proxy-server
pipeline = healthcheck cache tempauth proxy-server
```

```
[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true

[filter:swauth]
# the line below points to swauth as a separate project from swift
use = egg:swauth#swauth
set log_name = swauth
# Highly recommended to change this.
super_admin_key = swauthkey

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:cache]
use = egg:swift#memcache
memcache_servers = 192.168.0.1:11211

[filter:tempauth]
use = egg:swift#tempauth
user_test_tester = testing .admin
user_test2_tester2 = testing2 .admin
user_test_tester3 = testing3
user_system_root = testpass .admin .reseller_admin
```

Vorübergehend ist hier lediglich die `tempauth`-Authentifizierung aktiv. Dies sollte später auf *Keystone* geändert werden. Allerdings ist dieses aktuell noch in Entwicklung und noch nicht voll funktionsfähig.

Diese Konfiguration geht davon aus, dass `swift` über `https` genutzt wird, deshalb werden noch Zertifikate erzeugt:

```
# openssl req -new -x509 -nodes -out /etc/swift/cert.crt -keyout /etc/
swift/cert.key
```

Hiermit ist die Konfiguration des Proxy Servers abgeschlossen. Dies wird lediglich auf dem Host durchgeführt, auf dem auch die Komponente installiert wurde.

5.4.2 memcached

Swift nutzt `memcached` unter anderem zur Zwischenspeicherung von Abfragen und Tokens. Dieser muss selbstverständlich auch konfiguriert werden. Dieser Prozess ist nur auf dem Proxy Server notwendig.

Dafür tragen Sie die erreichbare IP-Adresse in der Datei `/etc/sysconfig/memcached` ein, der Eintrag `127.0.0.1` reicht nicht aus und wird ersetzt:

```
MEMCACHED_PARAMS="-d -l 192.168.0.1"
```

Danach starten Sie `memcached` neu und konfigurieren Sie dessen Aktivierung bei Systemstart:

```
# rcmemcached restart
# chkconfig memcached on
```

5.4.3 Account Server

Die Konfiguration des Account Servers findet in `/etc/swift/account-server.conf` statt. Diese muss auf beiden Hosts erfolgen, allerdings bietet es sich an, zuerst eine vollständige Konfiguration von *Swift* auf einem Host vorzunehmen und dann die entsprechenden Dateien auf jeden weiteren Host zu kopieren. Dabei muss die IP-Adresse unter `bind-ip` auf die Adresse des entsprechenden Hosts immer angepasst werden:

```
[DEFAULT]
bind_ip = 192.168.0.1
workers = 2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

5.4.4 Container Server

Für den Container Server gelten dieselben Vorgaben wie für den Account Server. Diese muss auf jedem beteiligten Host existieren, die IP-Adresse natürlich angepasst werden. Die Konfiguration findet in der Datei `/etc/swift/container-server.conf` statt:

```
[DEFAULT]
bind_ip = 192.168.0.1
workers = 2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]
```

5.4.5 Object Server

Abschließend muss noch der `Object Server` konfiguriert werden. Diese Konfiguration muss auch auf jedem beteiligten Host stattfinden und auch hier wird die IP-Adresse auf die des jeweiligen Hosts angepasst:

```
[DEFAULT]
bind_ip = 192.168.0.1
workers = 2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]
```

Damit ist die Konfiguration von *Swift* abgeschlossen. Die Dateien, die nun auf jedem Host vorhanden sein sollten, sind `account-server.conf`, `container-server.conf`, `object-server.conf`, `swift.conf` unterhalb von `/etc/swift` und die jeweils angepasste `/etc/rsyncd.conf`.

5.4.6 Ringe aufbauen

Die Ringe werden auf dem System erstellt, auf dem der `Proxy Server` betrieben werden soll. Nach der Erstellung kopieren Sie die einzelnen `gzip`-Archive in `/etc/swift/` auf alle anderen `swift`-Knoten.

Insgesamt werden drei Ringe benötigt, die Accounts, Container und Objekte enthalten werden.

Zuerst eine Übersicht des Befehls; ergänzen Sie die passenden Adressen entsprechend. Der für die Hosts im Beispiel notwendige Aufruf folgt später:

```
swift-ring-builder <builder_file> create <part_power> <replicas> <
  min_part_hours>
```

Hinweis: TCP-Ports

Achten Sie darauf, dass beim Hinzufügen eines Devices zu den einzelnen Ringen die richtigen TCP-Ports verwendet werden. Der `account-server` nutzt TCP Port-6002, der `container-server` TCP-Port 6001 und der `object-server` läuft auf TCP-Port 6000.

```
swift-ring-builder <builder_file> add z<zone>-<ip>:<port>/<device_name>
  _<meta> <wght>
```

```
swift-ring-builder account.builder add z<ZONE>-<STORAGE_LOCAL_NET_IP
>:6002/<DEVICE> 100
swift-ring-builder container.builder add z<ZONE>-<
STORAGE_LOCAL_NET_IP_1>:6001/<DEVICE> 100
swift-ring-builder object.builder add z<ZONE>-<STORAGE_LOCAL_NET_IP_1
>:6000/<DEVICE> 100
```

Für beide beteiligten Hosts erzeugen Sie mit folgendem Befehl die benötigten Objekte:

```
# swift-ring-builder account.builder create 18 2 1
# swift-ring-builder container.builder create 18 2 1
# swift-ring-builder object.builder create 18 2 1
```

Aufbau des Account-Ringes

Nach dem Erstellen des Accounts werden die beiden beteiligten Hosts hinzugefügt und der Inhalt des Accounts einmal dargestellt. Damit die Daten synchronisiert werden, führen Sie den initialen rebalance aus:

```
# swift-ring-builder account.builder add z1-192.168.0.1:6002/sda5 100
# swift-ring-builder account.builder add z2-192.168.0.2:6002/sda5 100

# swift-ring-builder account.builder

# swift-ring-builder account.builder rebalance
```

Aufbau des Container-Ringes

Für den Container müssen diesselben Schritte durchgeführt werden: Hosts hinzufügen und den Container aufbauen:

```
# swift-ring-builder container.builder add z1-192.168.0.1:6001/sda5 100
# swift-ring-builder container.builder add z2-192.168.0.2:6001/sda5 100

# swift-ring-builder container.builder

# swift-ring-builder container.builder rebalance
```

Aufbau des Object-Ringes

```
# swift-ring-builder object.builder add z1-192.168.0.1:6000/sda5 100
# swift-ring-builder object.builder add z2-192.168.0.2:6000/sda5 100

# swift-ring-builder object.builder

# swift-ring-builder object.builder rebalance
```

Die durch diesen Vorgang erzeugten Archive `account.ring.gz`, `container.ring.gz`, `object.ring.gz` kopieren Sie auf jeden Storageknoten in das Verzeichnis `/etc/swift`.

5.4.7 Der erste Start

Auf jedem beteiligten Host werden beim ersten Start die Dateiberechtigungen manuell gesetzt, dies ist danach nicht mehr nötig.

Der Befehl `swift-init` startet dann die konfigurierten Bestandteile auf dem entsprechenden Host:

```
# chown -R swift /srv/node/  
# chown -R swift:swift /etc/swift  
# swift-init all start
```

Für die einzelnen Dienste stehen passende Scripte bereit, die nach der Installation schon dafür sorgen, dass die *Swift*-Komponenten beim Systemstart mit gestartet werden. Diese rufen auch `swift-init` auf, welches dann die Komponenten startet:

```
# chkconfig | grep swift  
openstack-swift-account      on  
openstack-swift-container   on  
openstack-swift-object       on  
openstack-swift-proxy        on
```

5.4.8 Authentifizierung

Zum Zugriff auf den Object Store muss sich ein registrierter Benutzer authentifizieren. Der in der Konfiguration des Proxy Servers eingetragene `system:root`-Benutzer mit dem Passwort `testpass` wird für das Beziehen eines zeitlich befristeten Tokens genutzt. Sobald ein gültiges Token erhalten wurde, können die weiteren Tests ausgeführt werden.

5.5 Testen

Der erste Teil des Testes betrifft die Authentifizierung an *Swift*. Dazu holen Sie ein Token ein:

```
# curl -k -v -H 'X-Auth-Key: testpass' -H 'X-Auth-User: system:root'  
https://192.168.0.1:8080/auth/v1.0
```

Der Erhalt eines gültigen Tokens äussert sich wie folgt:

```
* About to connect() to 192.168.0.1 port 8080 (#0)  
* Trying 192.168.0.1... connected  
* Connected to 192.168.0.1 (192.168.0.1) port 8080 (#0)  
* successfully set certificate verify locations:  
* CAfile: none  
* CApath: /etc/ssl/certs/  
* SSLv3, TLS handshake, Client hello (1):  
* SSLv3, TLS handshake, Server hello (2):  
* SSLv3, TLS handshake, CERT (11):  
* SSLv3, TLS handshake, Server finished (14):  
* SSLv3, TLS handshake, Client key exchange (16):
```

```

* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using AES256-SHA
* Server certificate:
*   subject: /C=DE/ST=Bavaria/L=Location/O=Firma/OU=OpenStack
      Swift/CN=cn/emailAddress=mail@firma.de
*   start date: 2011-09-16 08:26:33 GMT
*   expire date: 2011-10-16 08:26:33 GMT
*   common name: cn (does not match '192.168.0.1')
*   issuer: /C=DE/ST=Bavaria/L=Location/O=Firma/OU=OpenStack Swift
      /CN=cn/emailAddress=mail@firma.de
*   SSL certificate verify result: self signed certificate (18),
      continuing anyway.
> GET /auth/v1.0 HTTP/1.1
> User-Agent: curl/7.19.0 (x86_64-suse-linux-gnu) libcurl/7.19.0
      OpenSSL/0.9.8h zlib/1.2.3 libidn/1.10
> Host: 192.168.0.1:8080
> Accept: */*
> X-Auth-Key: testpass
> X-Auth-User: system:root
>
< HTTP/1.1 200 OK
< X-Storage-Url: https://127.0.0.1:8080/v1/AUTH_system
< X-Storage-Token: AUTH_tka5bff789a9444b84ale34062cadb8df1
< X-Auth-Token: AUTH_tka5bff789a9444b84ale34062cadb8df1
< Content-Length: 0
< Date: Fri, 16 Sep 2011 14:30:08 GMT
<
* Connection #0 to host 192.168.0.1 left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):

```

Mit dem erhaltenen Token fragen Sie die Statistiken ab. Die Ausgabe wird sich unterscheiden, da im Testsystem hier bereits Daten liegen:

```

# swift -A https://192.168.0.1:8080/auth/v1.0 -U system:root -K
      testpass stat
      Account: AUTH_system
Containers: 2
      Objects: 5
      Bytes: 264082146
Accept-Ranges: bytes

```

Mit folgenden Befehl legen Sie eine Datei in das Zielverzeichnis `meinVerzeichnis`. Das Verzeichnis wird erstellt, falls es noch nicht existiert:

```

# swift -A http://192.168.0.1:8080/auth/v1.0 -U system:root -K testpass
      upload meinVerzeichnis <dateiname>

```

Den Inhalt des Verzeichnisses geben Sie mit folgendem Kommando aus:

```

# swift -A https://192.168.0.1:8080/auth/v1.0 -U system:root -K testing
      list meinVerzeichnis

```

Dort befindet sich nun die hochgeladene Datei.

5.6 Clients

Selbstverständlich können Sie auch mit anderen Clients auf den Object Store zugreifen. Allerdings ist es nicht möglich, den Store als `Blockdevice` zu mounten. Für MS Windows und Mac steht mit `Cyberduck` ein Client zur Verfügung, der nebenher auch weitere Protokolle unterstützt. Dieser findet sich unter <http://cyberduck.ch/>.

5.7 Glance für die Nutzung von Swift konfigurieren

Um `Glance` für die Nutzung von `Swift` zu konfigurieren, passen Sie einige Parameter in der Datei `/etc/glance/glance-api.conf` an:

```
default_store = swift

swift_store_auth_address = https://172.19.134.30:8080/auth/v1.0

swift_store_user = system:root

swift_store_key = testpass

swift_store_create_container_on_put = True
```

Alle anderen Optionen bleiben unverändert. Sollte es sich nun nicht um eine frische Installation von `Glance` handeln, löschen Sie die Datenbank zur Sicherheit einmal und legen Sie neu an:

```
# mysql -p
MySQL> drop database glance;
MySQL> create database glance;
MySQL> quit
```

Nach dem Neustart von `Glance` mit `glance-control all restart` wird das Datenbankschema neu erzeugt und mit dem bereits bekannten Befehl `glance-upload` werden nun die Images direkt in `Swift` gespeichert. `Nova` greift dann über `Glance` auf die Images zu.