Infrastructure as a Service mit OpenStack auf

SLES11 SP2 und openSUSE 12.1

Christian Berendt, André Nähring, Thomas Kärgel

16. April 2012

B1 Systems GmbH

Dieses Dokument bezieht sich auf OpenStack Essex sowie SLES11 SP2 und OpenSUSE 12.1.

Die aktuelle Version dieses Dokuments finden Sie hier:
`http://www.b1-systems.de/openstack`.

Mit Fragen und Anmerkungen oder um Fehler zu melden, wenden Sie sich an openstack@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.

Versionshistorie

VERSION	DATUM	GEÄNDERT WURDE	VON WEM
1.0	06.04.2012	Preview	Christian Berendt, André Nähring, Thomas Kärgel

Inhaltsverzeichnis

Einführung

In diesem Whitepaper wird die Installation und Grundkonfiguration von OpenStack auf SUSE Linux Enterprise Server SP2 und openSUSE 12.1 demonstriert. Nach einer kurzen Einführung zum Thema Cloud Computing und Hintergrundinformationen zum OpenStack-Projekt werden die einzelnen Komponenten von OpenStack der Reihe nach installiert und konfiguriert, so dass als Resultat eine voll funktionsfähige Infrastructure as a Service Umgebung zur Verfügung steht.

In den einzelnen Abschnitten werden lediglich Veränderungen an den Konfigurationsdateien hervorgehoben, die vollständigen Konfigurationsdateien befinden sich im Anhang.

Was ist Cloud Computing?

Bei Cloud Computing handelt es sich um ein Konzept, mit dem IT-Infrastruktur abstrahiert über ein Netzwerk angeboten werden kann. Dabei kann es sich zum Beispiel um Datenspeicher oder CPU-Leistung handeln. Angebotene Ressourcen können zur Laufzeit je nach Bedarf dynamisch erweitert oder reduziert werden. Dies soll mit Minimalaufwand und ohne Interaktion mit einem Service Provider erfolgen können.

Die offizielle Beschreibung von Cloud Computing des NIST (National Institute of Standards and Technology) beinhaltet fünf Eigenschaften (*Essential Characteristics*), drei Servicemodelle (*Service Models*) und vier Liefermodelle (*Deployment Models*). Detailierte Informationen zu dieser Definition finden sich unter `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.

Eigenschaften

On-demand Self-service Ein Nutzer kann ohne weitere Interaktion mit einem Service Provider die Eigenschaften der Systeme ändern.

Broad Network Access Die Ressourcen sind mit Hilfe von Standardmechanismen über ein Netzwerk verfügbar.

Ressource Pooling Die Ressourcen des Anbieters werden für mehrere Kunden zusammengefasst, allerdings mit physischen und virtuellen Ressourcen, die den Kunden dynamisch zur Verfügung gestellt werden.

Rapid Elasticity Ressourcen können erweitert oder freigegeben werden. Dies kann unter Umständen auch automatisch geschehen, damit schnell steigende Anforderungen automatisch bedient werden können. Für den Nutzer erscheinen Ressourcen unlimitiert.

Measured Service Die Nutzung von Ressourcen kann gemessen, kontrolliert und ausgewertet werden. Zusätzlich sollen Ressourcen durch das Cloud-System automatisch kontrolliert und optimiert werden.

Servicemodelle

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.

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.

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

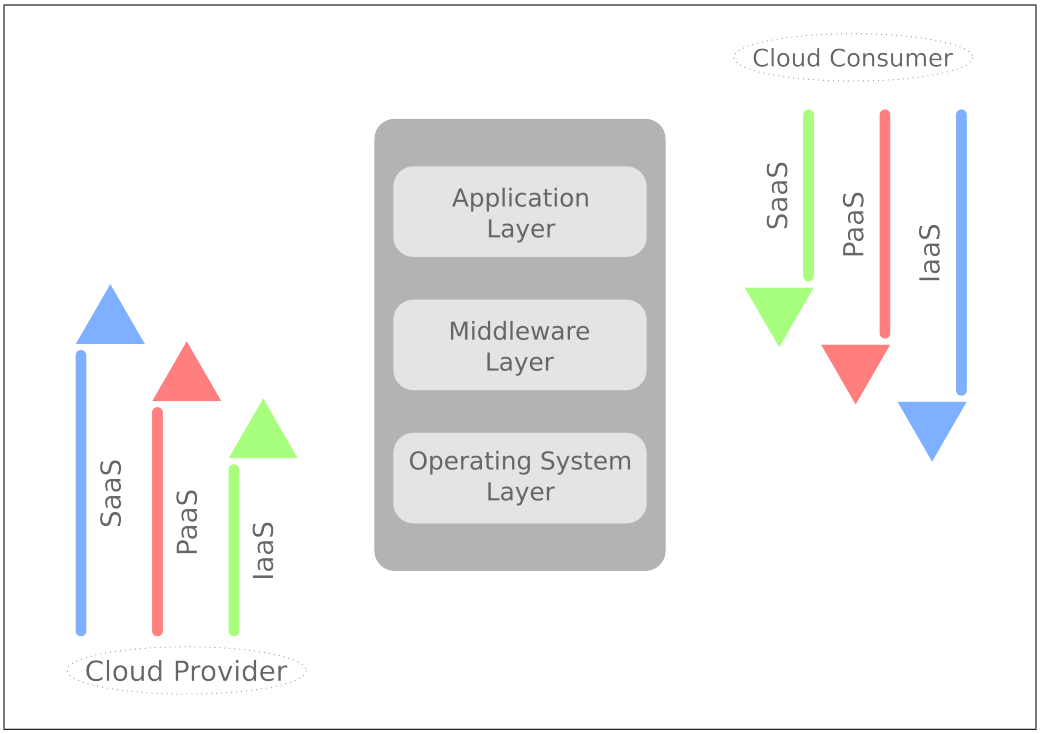

Abbildung 1: Zuordnung der einzelnen Servicemodelle und der entsprechenden abstrahierten Ebenen

Liefermodelle

Private Cloud Die Infrastruktur der Cloud wird nur einem Kunden zur Verfügung gestellt oder gehört diesem.

Community Cloud Diese Cloud wird exklusiv für eine Community zur Verfügung gestellt.

Public Cloud Die Ressourcen dieser Cloud werden für die breite Öffentlichkeit vergeben.

Hybrid Cloud Dabei handelt es sich um eine Mischung aus einer Private, Community oder einer Public Cloud.

Was ist OpenStack?

OpenStack bildet eine IaaS-Cloud ab. Dabei ist das Ziel des Projektes, frei übersetzt, die universelle Open Source Cloud Computing Platform für private und öffentliche Clouds zu werden.
Es existieren mehrere Kernprojekte, aus denen OpenStack besteht. Jedes davon kann einzeln eingesetzt und einfach implementiert werden.
Der gesamte Code des OpenStack-Projektes steht unter der Apache 2.0 Lizenz und ist über das Internet frei verfügbar.

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 begann man mit den Veröffentlichungen der einzelnen Versionen:

VERSION	ERSCHEINUNGSMONAT
Austin	Oktober 2010
Bexar	Februar 2011
Cactus	April 2011
Diablo	September 2011
Essex	April 2012

Komponenten von OpenStack

OpenStack besteht aus Komponenten, die allesamt auch allein und eigenständig betrieben werden können. Die folgende Auflistung stellt die aktuellen Kernprojekte in der Reihenfolge ihrer Erscheinung vor:

swift auch bekannt als OpenStack Object Storage. Hierbei handelt es sich um einen Objectstore, ähnlich Amazons S3. Über eine API können Daten hinterlegt und verwaltet werden. Dieser Storage kann nicht als Block Storage genutzt oder gemountet werden.

nova OpenStack Compute. Dies ist die Komponente zur Erstellung und Verwaltung von virtuellen Maschinen.

glance OpenStack Image Service, dient zur Verwaltung von Imagedateien. Diese werden an einem nahezu beliebigen Speicherort und -typ abgelegt und über glance addressiert.

horizon ehemals das Dashboard. Eine Weboberfläche zur Verwaltung der einzelnen OpenStack-Komponenten.

keystone das Identitätsmanagement dient zur Authentifizierung. Dabei können unterschiedliche Backends angebunden und genutzt werden.

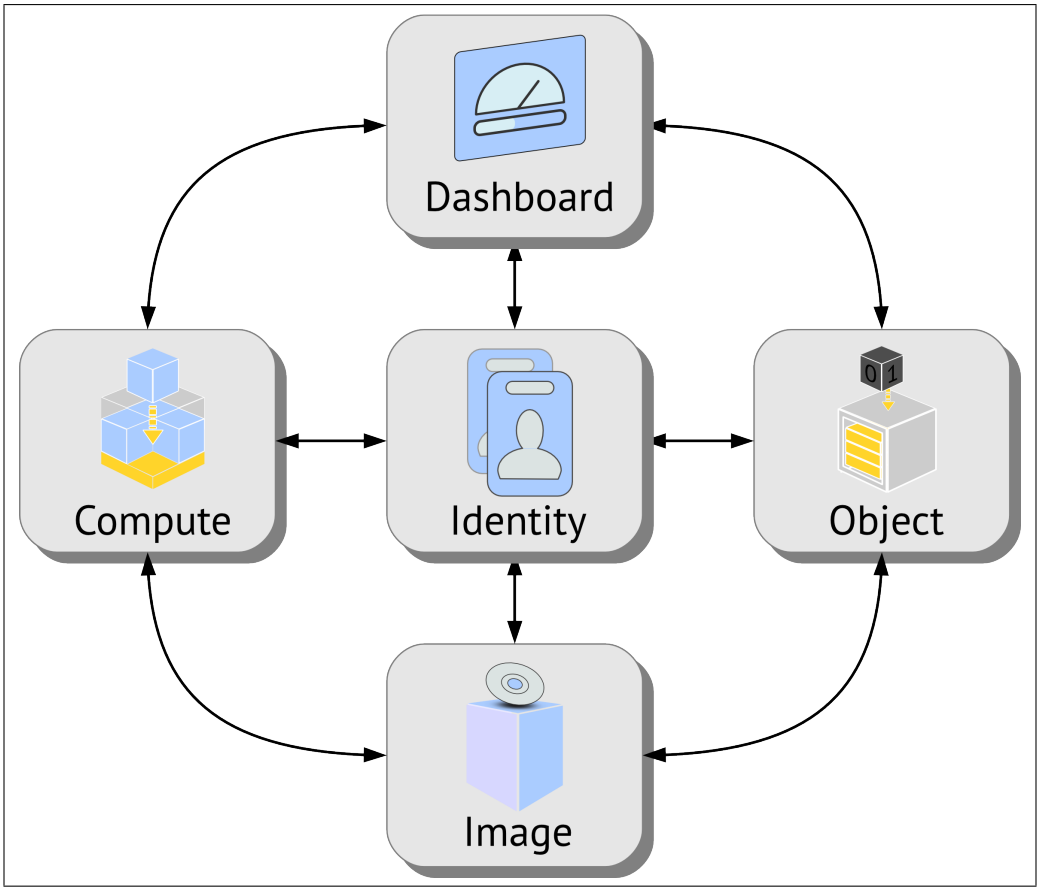

Abbildung 2: Beziehungen der Komponenten zueinander

Repositories

Zur fehlerfreien Nutzung von externen Paketquellen müssen deren Public Keys zunächst importiert werden. Jedes System, auf dem Pakete von OpenStack durch das von B1 Systems bereitgestellte Repository installiert werden sollen, muß über den entsprechenden Schlüssel verfügen. Der Key Import kann direkt über einen Aufruf von `rpm` erledigt werden. Nachfolgend der entsprechende Befehl für das Repository von RabbitMQ Server:

```
# rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/tools:/rabbitmq/SLE_11_SP2/repodata/repomd.xml
    .key
```

Die von B1 Systems zur Verfügung gestellten Projekte zu OpenStack befinden sich auf dem Open Build Service von openSUSE (`http://build.opensuse.org`) unterhalb des Projektes `isv:B1-Systems:Openstack`.

Essex

Für den letzten Release `Essex` finden sich Pakete im Projekt `isv:B1-Systems:OpenStack:release:Essex`. Notwendige Abhängigkeiten für diese Pakete liegen im Projekt `isv:B1-Systems:OpenStack:release:Essex:requirements`.

Im Projekt `isv:B1-Systems:OpenStack:tools:rabbitmq` steht eine aktuelle Version von RabbitMQ zur Verfügung.

Nachfolgend werden diese Repositories genutzt.

Development

Zusätzlich liegen die Pakete mit aktuellen Entwicklungsversionen von OpenStack im Projekt `isv:B1-Systems:OpenStack:devel`; deren Abhängigkeiten im Projekt `isv:B1-Systems:OpenStack:requirements`. Die Snapshots werden für gewöhnlich jeden Tag aktualisiert.

SUSE Linux Enterprise Server 11 SP2

Für SLES11 SP2 werden die nachfolgenden Kommandos einmalig auf jedem System ausgeführt. Sie legen fest, welche Pakete aus den Repositories verwendet werden:

```
# rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/release:/Essex/SLE_11_SP2/repodata/repomd.xml.
    key

# zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems
    :/OpenStack:/release:/Essex/SLE_11_SP2/isv:B1-Systems:OpenStack:
    release:Essex.repo
```

```
Adding repository 'OpenStack Essex Release (SLE_11_SP2)' [done]
Repository 'OpenStack Essex Release (SLE_11_SP2)' successfully added
Enabled: Yes
Autorefresh: No
GPG check: Yes
URI: http://download.opensuse.org/repositories/isv:/B1-Systems:/
    OpenStack:/release:/Essex/SLE_11_SP2/
```

```
# rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/release:/Essex:/requirements/SLE_11_SP2/
    repodata/repomd.xml.key

# zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems
    :/OpenStack:/release:/Essex:/requirements/SLE_11_SP2/isv:B1-
    Systems:OpenStack:release:Essex:requirements.repo
Adding repository 'requirements for the packages of the OpenStack
    Essex release (SLE_11_SP2)' [done]
Repository 'requirements for the packages of the OpenStack Essex
    release (SLE_11_SP2)' successfully added
Enabled: Yes
Autorefresh: No
GPG check: Yes
URI: http://download.opensuse.org/repositories/isv:/B1-Systems:/
    OpenStack:/release:/Essex:/requirements/SLE_11_SP2/
```

Durch einen Aufruf von `zypper lr` können die installierten Repositories überprüft werden, die Liste sollte nun (mit abweichenden IDs) folgende Einträge aufweisen:

```
 9 | isv_B1-Systems_OpenStack_release_Essex                          |
     OpenStack Essex Release (SLE_11_SP2)
                                          | Ja        | Nein
10 | isv_B1-Systems_OpenStack_release_Essex_requirements             |
    requirements for the packages of the OpenStack Essex release (
    SLE_11_SP2) | Ja        | Nein
```

openSUSE 12.1

Für openSUSE 12.1 werden die nachfolgenden Kommandos einmalig auf jedem System ausgeführt. Sie legen fest, welche Pakete aus den Repositories verwendet werden:

```
# rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/release:/Essex:/requirements/openSUSE_12.1/
    repodata/repomd.xml.key

# zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems
    :/OpenStack:/release:/Essex:/requirements/openSUSE_12.1/isv:B1-
    Systems:OpenStack:release:Essex:requirements.repo
```

```
Adding repository 'requirements for the packages of the OpenStack
    Essex release (openSUSE_12.1)' [done]
Repository 'requirements for the packages of the OpenStack Essex
    release (openSUSE_12.1)' successfully added
Enabled: Yes
Autorefresh: No
GPG check: Yes
URI: http://download.opensuse.org/repositories/isv:/B1-Systems:/
    OpenStack:/release:/Essex:/requirements/openSUSE_12.1/
```

```
# rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/release:/Essex/openSUSE_12.1/repodata/repomd.
    xml.key

# zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems
    :/OpenStack:/release:/Essex/openSUSE_12.1/isv:B1-Systems:OpenStack
    :release:Essex.repo
Adding repository 'OpenStack Essex Release (openSUSE_12.1)' [done]
Repository 'OpenStack Essex Release (openSUSE_12.1)' successfully
    added
Enabled: Yes
Autorefresh: No
GPG check: Yes
URI: http://download.opensuse.org/repositories/isv:/B1-Systems:/
    OpenStack:/release:/Essex/openSUSE_12.1/
```

Vorbereitung

Wir setzen voraus, dass für die Installation der einzelnen Dienste bereits Systeme zur Verfügung stehen. Es ist ausreichend, eine minimale Installation von SLES11 SP2 bzw. openSUSE 12.1 vorzubereiten. Als Storage sollten je System 25 GByte für das Betriebssystem zur Verfügung stehen.

Systemübersicht

Für einen Proof of Concept empfehlen wir die in diesem Dokument vorgestellte Systemlandschaft.

Generell lassen sich alle Dienste, ausgenommen der Nova Compute Nodes, für erste Teststellungen auf einem System betreiben.

Wir empfehlen für Nova Compute Nodes ausschließlich bare-metal Systeme einzusetzen und nicht mit QEMU innerhalb einer virtuellen Maschine (VM) zu arbeiten.

HOSTNAME	IP-ADRESSE	DIENST	TYP	SIZING
database	172.19.111.110	Datenbank	virtuell	4 VCPUs, 16 GByte Memory
rabbitmq	172.19.111.111	Queueing	virtuell	4 VCPUs, 16 GByte Memory
keystone	172.19.111.112	Keystone	virtuell	2 VCPUs, 4 GByte Memory
glance-api	172.19.111.113	Glance API	virtuell	2 VCPUs, 4 GByte Memory + 100 GByte Storage
glance-registry	172.19.111.114	Glance Registry	virtuell	2 VCPUs, 4 GByte Memory
nova-api	172.19.111.115	Nova API	virtuell	2 VCPUs, 4 GByte Memory
nova-scheduler	172.19.111.116	Nova Scheduler	virtuell	2 VCPUs, 4 GByte Memory
nova-volume	172.19.111.117	Nova Volume	virtuell	2 VCPUs, 4 GByte Memory + 100 GByte Storage
horizon	172.19.111.118	Horizon	virtuell	2 VCPUs, 4 GByte Memory
nova-compute-001	172.19.111.120	Nova Compute	bare-metal	2x Quadcore, 32 GByte RAM, 1 TByte Storage
nova-compute-002	172.19.111.121	Nova Compute	bare-metal	2x Quadcore, 32 GByte RAM, 1 TByte Storage

Als Domain verwendet wird `openstack.b1-systems.de`, damit ergeben sich Hostnames wie `database.openstack.b1-systems.de` oder `keystone.openstack.b1-systems.de`.

Anmerkungen

SUSE Linux Enterprise Server SP2

ACHTUNG: In dem mit SLES11 SP2 ausgeliefertem Python Interpreter sind derzeitig Fehler enthalten. Dadurch ist es momentan nicht möglich, OpenStack unter SLES11 SP2 zu betreiben. Bitte kontaktieren Sie uns unter `openstack@b1-systems.de`, um ein fehlerbereinigtes Paket für Python zu bekommen.

Mit nachfolgenden Python-Skript kann überprüft werden, ob die Fehler bereits durch ein Update von SUSE behoben wurden:

```
#!/usr/bin/python

## vim: set syn=on ts=4 sw=4 sts=0 noet foldmethod=indent:
## purpose: check if python interpreter contains blocking issues for
   OpenStack
## copyright: B1 Systems GmbH <info@b1-systems.de>, 2012.
## license: GPLv3+, http://www.gnu.org/licenses/gpl-3.0.html
## author: Christian Berendt <berendt@b1-systems.de>, 2012.

print "test if keyword arguments with unicode keys are working
   without problems"
print "expected result: {u'key': 'value'}"
print

def testing(**kwargs):
    print kwargs

try:
    testing(**{u'key': 'value'})
except TypeError:
    print "ERROR: catched TypeError, keyword arguments with unicode
       keys don't work"


print
print "test if ident of thread is working without problems"
print "expected result: a long integer (for example: 140630265239296)
   "
print

import threading
t = threading.current_thread()
ident = t.ident
if ident == None:
    print "ERROR: ident of current thread is None, ident of threads
       don't work"
else:
```

```
    print ident
```

Dieses Skript wird per `chmod +x` ausführbar gemacht und dann mit dem gewählten Namen aufgerufen:

```
./testing.py
test if keyword arguments with unicode keys are working without
   problems
expected result: {u'key': 'value'}

{u'key': 'value'}

test if ident of thread is working without problems
expected result: a long integer (for example: 140630265239296)

140614453929728
```

Sollte die installierte Python-Version fehlerhaft sein, erscheint folgende Ausgabe:

```
test if keyword arguments with unicode keys are working without
   problems
expected result: {u'key': 'value'}

ERROR: catched TypeError, keyword arguments with unicode keys are don
   't work

test if ident of thread is working without problems
expected result: a long integer (for example: 140630265239296)

ERROR: ident of current thread is None, ident of threads don't work
```

In diesem Fall ist es erforderlich Python zu aktualisieren, damit eine Nutzung von OpenStack möglich ist.

openSUSE 12.1

Wurde `openSUSE 12.1 minimal` installiert, muss zunächst das Paket `patterns-openSUSE-minimal_base-conflicts` deinstalliert werden, da sonst die Installation von Python nicht möglich ist.

```
# zypper remove -y patterns-openSUSE-minimal_base-conflicts
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following package is going to be REMOVED:
  patterns-openSUSE-minimal_base-conflicts

1 package to remove.
```

```
After the operation, 1.0 KiB will be freed.
Continue? [y/n/?] (y): y
Removing patterns-openSUSE-minimal_base-conflicts-12.1-25.21.1 [done]
```

Noch fehlende Komponenten

Derzeit werden in diesem Dokument die Komponenten *Keystone*, *Glance*, *Nova* sowie *Horizon* behandelt.

Einen Abschnitt über Swift werden wir demnächst ergänzen, sowie einen Ausblick auf die Verwendung von Quantum und Melange mit aufnehmen.

Datenbank

Die Nutzdaten der einzelnen OpenStack-Dienste werden in einer SQL-Datenbank gespeichert, wobei jede Komponente ihre Daten in einer eigenen Datenbank vorhält.

Standardmäßig werden SQLite-Dateien direkt auf dem lokalem Dateisystem genutzt. In diesem Dokument werden wir die Verwendung von MySQL beschreiben, welche wir momentan für den produktiven Einsatz empfehlen.

MySQL

SUSE Linux Enterprise Server SP2

Unter SLES11 SP2 das Paket `mysql` installieren:

```
# zypper install mysql
```

openSUSE 12.1

Unter openSUSE 12.1 das Paket `mysql-community-server` installieren:

```
# zypper install mysql-community-server
```

Installation des Schemas

Zunächst wird der Dienst `mysql` aktiviert und gestartet:

```
# insserv mysql
# service mysql start
```

Beim ersten Start werden die von MySQL selbst genutzten Schema-Informationen installiert. Anschließend wird das Passwort für den Superuser gesetzt:

```
# /usr/bin/mysqladmin -u root password "somepassword"
```

Anschließend wird die Datei `/root/openstack-grants.sql` mit nachfolgendem Inhalt erstellt. Der Einfachheit halber erlauben wir den Zugriff auf die einzelnen Datenbanken von jedem System aus:

```
CREATE DATABASE keystone; GRANT ALL ON keystone.* TO 'keystone'@'%'
    IDENTIFIED BY 'somepassword';
CREATE DATABASE glance; GRANT ALL ON glance.* TO 'glance'@'%'
    IDENTIFIED BY 'somepassword';
CREATE DATABASE nova; GRANT ALL ON nova.* TO 'nova'@'%' IDENTIFIED BY
     'somepassword';
```

Durch nachfolgenden Aufruf werden die benötigten Datenbanken sowie Benutzer erstellt:

```
# mysql -u root -p < /root/openstack-grants.sql
```

Wenn alles funktioniert hat, sollte das Listing vorhandener Datenbanken wie folgt aussehen:

```
database:~ # mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.5.16-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights
    reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
    statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| glance             |
| horizon            |
| keystone           |
| mysql              |
| nova               |
| performance_schema |
| quantum            |
| test               |
+--------------------+
10 rows in set (0.00 sec)
```

Die Datenbanken werden bei Installation und Konfiguration der einzelnen Komponenten initialisiert.

Queueing

Die einzelnen Dienste von OpenStack kommunizieren über das *Advanced Message Queueing Protocol* (AMQP) untereinander. Diese Funktionalität wird von einem weiteren Serverdienst bereitgestellt. In diesem Dokument werden wir die Verwendung eines RabbitMQ Servers beschreiben. Diesen stellen wir in einer aktuellen Version im Projekt `isv:B1-Systems:OpenStack:tools:rabbitmq` zur Verfügung.

RabbitMQ

Repositories für SLES11 SP2

```
rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/tools:/rabbitmq/SLE_11_SP2/repodata/repomd.xml
    .key
zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems:/
    OpenStack:/tools:/rabbitmq/SLE_11_SP2/isv:B1-Systems:OpenStack:
    tools:rabbitmq.repo
```

Repositories für openSUSE 12.1

```
# rpm --import http://download.opensuse.org/repositories/isv:/B1-
    Systems:/OpenStack:/tools:/rabbitmq/openSUSE_12.1/repodata/repomd.
    xml.key
# zypper ar http://download.opensuse.org/repositories/isv:/B1-Systems
    :/OpenStack:/tools:/rabbitmq/openSUSE_12.1/isv:B1-Systems:
    OpenStack:tools:rabbitmq.repo
```

Installation

Zur Installation von RabbitMQ unter SUSE Linux Enterprise Server SP2 sowie openSUSE 12.1 ist folgender Aufruf ausreichend:

```
# zypper install rabbitmq-server
```

Nun wird noch das Starten des Dienstes beim Start des Servers aktiviert und der Dienst erstmals gestartet:

```
# chkconfig rabbitmq-server on
# service rabbitmq-server start
Starting rabbitmq-server: SUCCESS
rabbitmq-server.
```

Abschließend wird das Passwort für den Account `guest` gesetzt, welcher später von den einzelnen Diensten in diesem Dokument genutzt wird:

```
# rabbitmqctl change_password guest 'somepassword'
Changing password for user "guest" ...
...done.
```

Administration

Um den Status eines aktuell laufenden RabbitMQ-Servers abzufragen oder andere Signale an den Server zu senden, wird das Kommando `rabbitmqctl` genutzt. Dieses verfügt über verschiedene Parameter und dient zur Verwaltung des Servers:

```
# rabbitmqctl status
Status of node rabbit@database ...
[{pid,12535},
 {running_applications,[{rabbit,"RabbitMQ","2.8.1"},
                        {mnesia,"MNESIA  CXC 138 12","4.6"},
                        {os_mon,"CPO  CXC 138 46","2.2.8"},
                        {sasl,"SASL  CXC 138 11","2.2"},
                        {stdlib,"ERTS  CXC 138 10","1.18"},
                        {kernel,"ERTS  CXC 138 10","2.15"}]},
 {os,{unix,linux}},
 {erlang_version,"Erlang R15B (erts-5.9) [source] [64-bit] [async-
    threads:30] [hipe] [kernel-poll:true]\n"},
 {memory,[{total,26375416},
          {processes,9975284},
          {processes_used,9975270},
          {system,16400132},
          {atom,504409},
          {atom_used,477154},
          {binary,743584},
          {code,11999393},
          {ets,741736}]},
 {vm_memory_high_watermark,0.3999999996312298},
 {vm_memory_limit,433874534},
 {file_descriptors,[{total_limit,924},
                    {total_used,3},
                    {sockets_limit,829},
                    {sockets_used,1}]},
 {processes,[{limit,1048576},{used,114}]},
 {run_queue,0},
 {uptime,289}]
...done.
```

RabbitMQ stellt mehrere VHosts zur Verfügung, über die einzelne Clients kommunizieren können. Den Status der VHosts erfragt man mit dem folgenden Kommando:

```
# rabbitmqctl list_vhosts
Listing vhosts ...
```

```
/
...done.
```

Analog dazu existiert eine Möglichkeit, die aktuellen Benutzer abzufragen:

```
# rabbitmqctl list_users
Listing users ...
guest   [administrator]
...done.
```

Zur Kontrolle, ob ein RabbitMQ-Server lauscht, kann auf `netstat` zurückgegriffen werden. Im folgenden Beispiel werden die entsprechenden RabbitMQ-Dienste herausgefiltert:

```
# netstat -tulpen | grep -e beam -e epmd
tcp        0      0 0.0.0.0:5672            0.0.0.0:*
    LISTEN      104        28650      12535/beam
tcp        0      0 0.0.0.0:4369            0.0.0.0:*
    LISTEN      104        28330      12515/epmd
tcp        0      0 0.0.0.0:59488           0.0.0.0:*
    LISTEN      104        28621      12535/beam
```

Wird ein Problem mit dem RabbitMQ-Server vermutet, kann dieses in den Logdateien des Daemons kontrolliert werden. Diese befinden sich unterhalb des Verzeichnisses `/var/log/rabbitmq`.

Keystone

Der Identity Service *Keystone* übernimmt die zentrale Verwaltung von Kunden/Projekten, Benutzern sowie Rollen und stellt einen Katalog aller in einer OpenStack Umgebung vorhandenen Dienste bereit. Dadurch kann *Keystone* als zentraler Einstiegspunkt in die Umgebung genutzt werden. Weiterhin übernimmt er die Authentifizierung von Benutzern durch Bereitstellung eines zeitlich begrenzt nutzbaren Tokens nach erfolgtem Login.

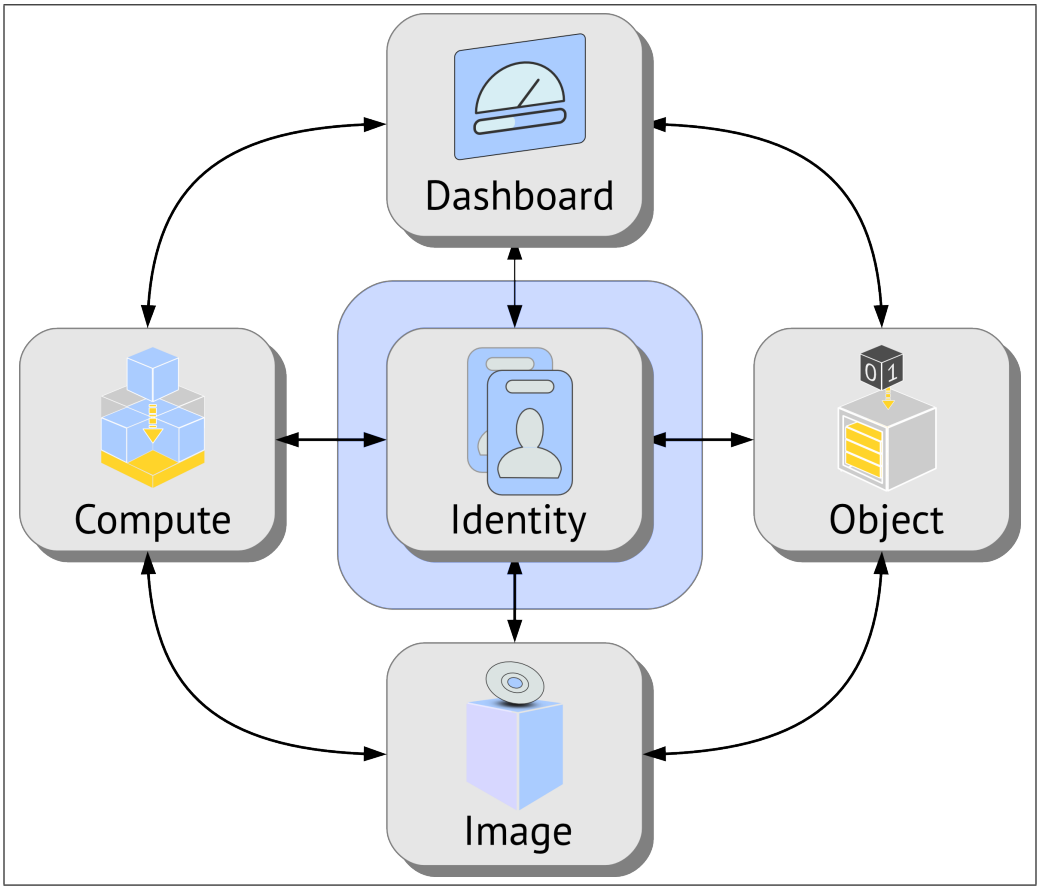

Abbildung 3: Beziehung anderer Komponenten zu *Keystone*

In *Keystone* werden Nutzdaten folgender Kategorien verwaltet:

User Benutzer

Tenants Kunden/Projekte

Services einzelne Dienste, z.B. *Glance*

Role Rollen für Benutzer

Weiterhin werden die Beziehungen zwischen Benutzern und Kunden/Projekt sowie Benutzern und Rollen hinterlegt. Als Backend kann neben einer Datenbank auch ein Key-Value-Storage oder LDAP verwendet werden.
Detaillierte Informationen zu Keystone finden sich unter `http://keystone.openstack.org`.

Architektur

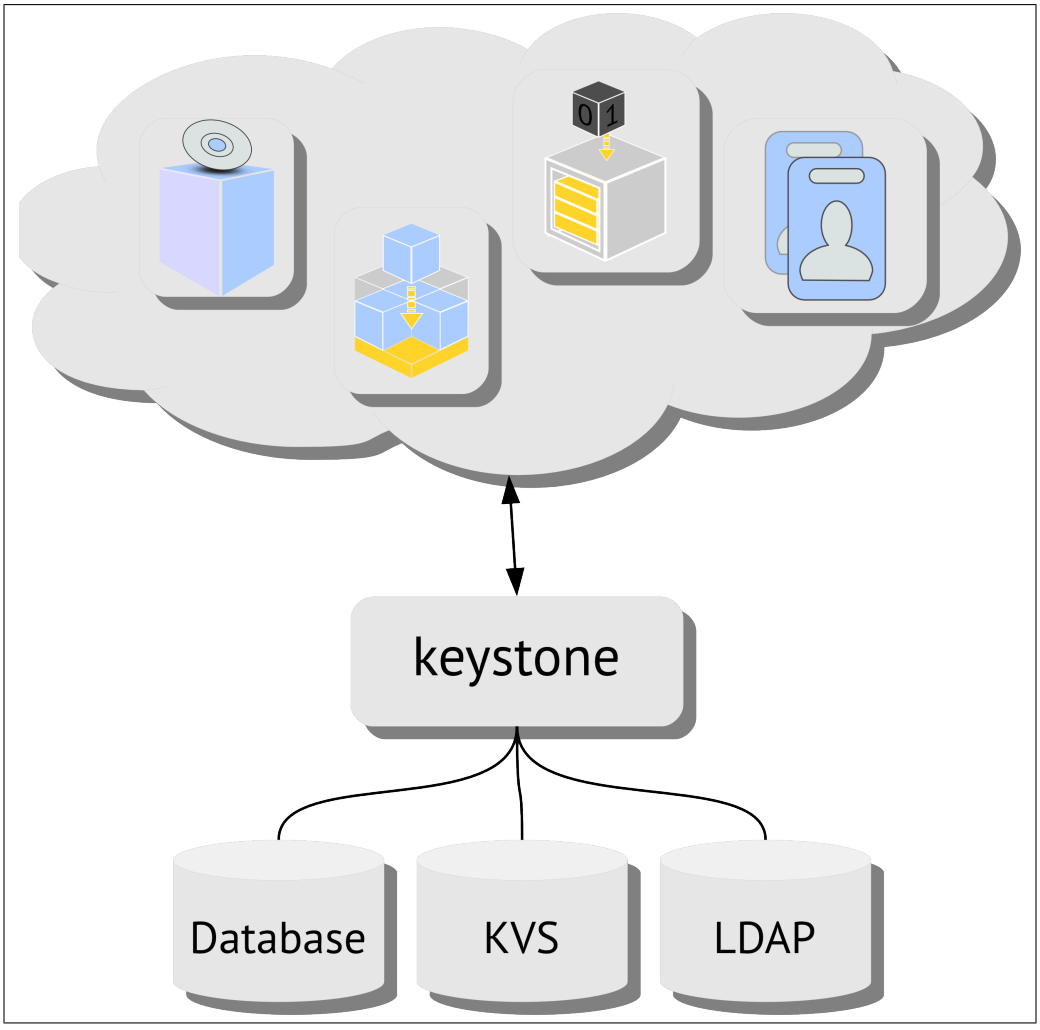

Abbildung 4: Keystone Architektur

Keystone besteht momentan aus einem einzelnem Dienst.

Installation

Die Installation von Keystone erfolgt über das Installieren der entsprechenden Pakete:

```
# zypper install openstack-keystone
# zypper install openstack-keystoneclient
```

Konfiguration

Die Konfiguration von *Keystone* findet in den Konfigurationsdateien unterhalb von `/etc/keystone` statt.

Servicekatalog

Die Datei `/etc/keystone/default_catalog.templates` beschreibt die einzelnen Regionen für die unterschiedlichen Dienste. Hier müssen für die einzelnen Dienste die entsprechenden Servernamen gesetzt werden.

In der Vorlage reicht, sollten alle Dienste auf demselben Host laufen, die unveränderte Konfiguration aus:

```
catalog.$NAME_OF_REGION.$NAME_OF_SERVICE.publicURL = http://$FQDN:
    $PORT/$PROTOCOL_VERSION/
catalog.$NAME_OF_REGION.$NAME_OF_SERVICE.adminURL = http://$FQDN:
    $PORT/$PROTOCOL_VERSION/
catalog.$NAME_OF_REGION.$NAME_OF_SERVICE.internalURL = http://$FQDN:
    $PORT/$PROTOCOL_VERSION/
catalog.$NAME_OF_REGION.$NAME_OF_SERVICE.name = $NAME
```

Es ist möglich, auf Werte aus `/etc/keystone/keystone.conf` oder auf Werte, die sich aus Anfragen an *Keystone* ergeben, zuzugreifen. So wird z.B. `$(tenant_id)s` durch die Kunden-ID der aktuellen Anfrage ersetzt oder `$(public_port)s` durch den Parameter `public_port` aus der Sektion `DEFAULT` aus `/etc/keystone/keystone.conf` ersetzt.

Ein Eintrag für die API von Compute in der Region `RegionOne` würde wie folgt aussehen (vollständiges Beispiel im Anhang):

```
catalog.RegionOne.compute.publicURL = http://nova-api.openstack.b1-
    systems.de:$(compute_port)s/v1.1/$(tenant_id)s
catalog.RegionOne.compute.adminURL = http://nova-api.openstack.b1-
    systems.de:$(compute_port)s/v1.1/$(tenant_id)s
catalog.RegionOne.compute.internalURL = http://nova-api.openstack.b1-
    systems.de:$(compute_port)s/v1.1/$(tenant_id)s
catalog.RegionOne.compute.name = Compute Service
```

Weiterhin muss in der Konfiguration `/etc/keystone/keystone.conf` entsprechend auf diese Datei verwiesen werden:

```
[catalog]
driver = keystone.catalog.backends.templated.TemplatedCatalog
template_file = /etc/keystone/default_catalog.templates
```

Die Konfiguration der Endpunkte kann direkt in der Datenbank hinterlegt und über die `keystone` CLI zu verwaltet werden. Hier zu wird der Parameter `driver` in der Sektion `catalog` angepasst. Der Parameter `template_file` ist dann nicht mehr erforderlich.

Logging

Das Loggingverhalten von Keystone wird in der Datei `/etc/keystone/logging.conf` konfiguriert. Dazu kopiert man einfach die mit dem Paket gelieferte Datei `/etc/keystone/logging.conf.sample` nach `/etc/keystone/logging.conf`.

Anschliessend wird die Konfiguration in `/etc/keystone/keystone.conf` in der Sektion `[DEFAULT]` im Paramter `log_config` hinterlegt. Die Logdatei selbst wird über den Parameter `log_file` festgelegt:

```
log_config = /etc/keystone/logging.conf
log_file = /var/log/keystone/keystone.log
```

Datenbank

```
[sql]
connection = mysql://keystone:testing@database.openstack.b1-systems.
   de/keystone
```

```
[identity]
driver = keystone.identity.backends.sql.Identity
```

```
[token]
driver = keystone.token.backends.sql.Token
```

```
[ec2]
driver = keystone.contrib.ec2.backends.sql.Ec2
```

Um die vorab erstelle Datenbank mit dem entsprechenden Schema für *Keystone* zu füllen, wird die Management CLI `keystone-manage` mit dem passenden Parameter aufgerufen:

```
# keystone-manage db_sync
```

Auf die Verwendung von LDAP oder einem KVS (Key-Value-Storage) gehen wir in diesem Dokument nicht näher ein.

Admintoken

Das Admintoken wird zur Anbindung der anderen OpenStack-Komponenten genutzt, sowie zur initialen Erstellung von Kunden/Projekten, Rollen und Benutzern.

Aktuell wird es fest in der `/etc/keystone/keystone.conf` hinterlegt:

```
admin_token = testing
```

Rollen-Rechte-Zuordnung

Mit Hilfe der Regeln in `policy.json`, bei *Keystone* die Datei `/etc/keystone/policy.json`, können derzeitig in *Keystone* hinterlegte Rollen auf Funktionen innerhalb einzelner Dienste gelegt werden. Diese Dateien finden sich auch bei Glance und Nova.

Standardmäßig können alle Benutzer, die der Rolle `admin` zugewiesen sind, auf *Keystone* als Admin zugreifen. Dieses Verhalten kann über diese Datei verändert werden, falls das notwendig wird.

```
{
    "admin_required": [["role:admin"], ["is_admin:1"]]
}
```

Start

Der Dienst `openstack-keystone` wird so eingestellt, dass er beim Start des Systems automatisch gestartet wird. Der erste Start erfolgt hier manuell:

```
# insserv openstack-keystone
# service openstack-keystone start
```

Benutzung

Bezug eines Tokens

Nach erfolgreicher Kommunikation mit *Keystone* erhält der nun authentifizierte Benutzer ein Token, welches für alle weiteren Operationen genutzt werden kann. Da dieses Token zeitlich befristet ist, ist es notwendig, nach Ablauf des Tokens ein neues anzufordern. Dazu dient folgendes Kommando:

```
export TOKEN=$(curl -s -d  '{"auth":{"passwordCredentials": {"
    username": "admin", "password": "testing"}, "tenantName": "
    b1systems"}}' -H "Content-type: application/json" http://localhost
    :5000/v2.0/tokens | python -c "import sys; import json; tok = json
    .loads(sys.stdin.read()); print tok['access']['token']['id'];")
```

Damit steht ein Token in der Umgebungsvariablen `TOKEN` zur Verfügung. Diese wird später benötigt.

Mittlerweile ist *Keystone* als Authentifizierungsdienst in allen CLIs der Komponenten enthalten und die Credentials (Kunde/Projekt, Benutzer, Passwort) können direkt beim Aufruf der entsprechenden CLI mit angegeben werden.

Glance

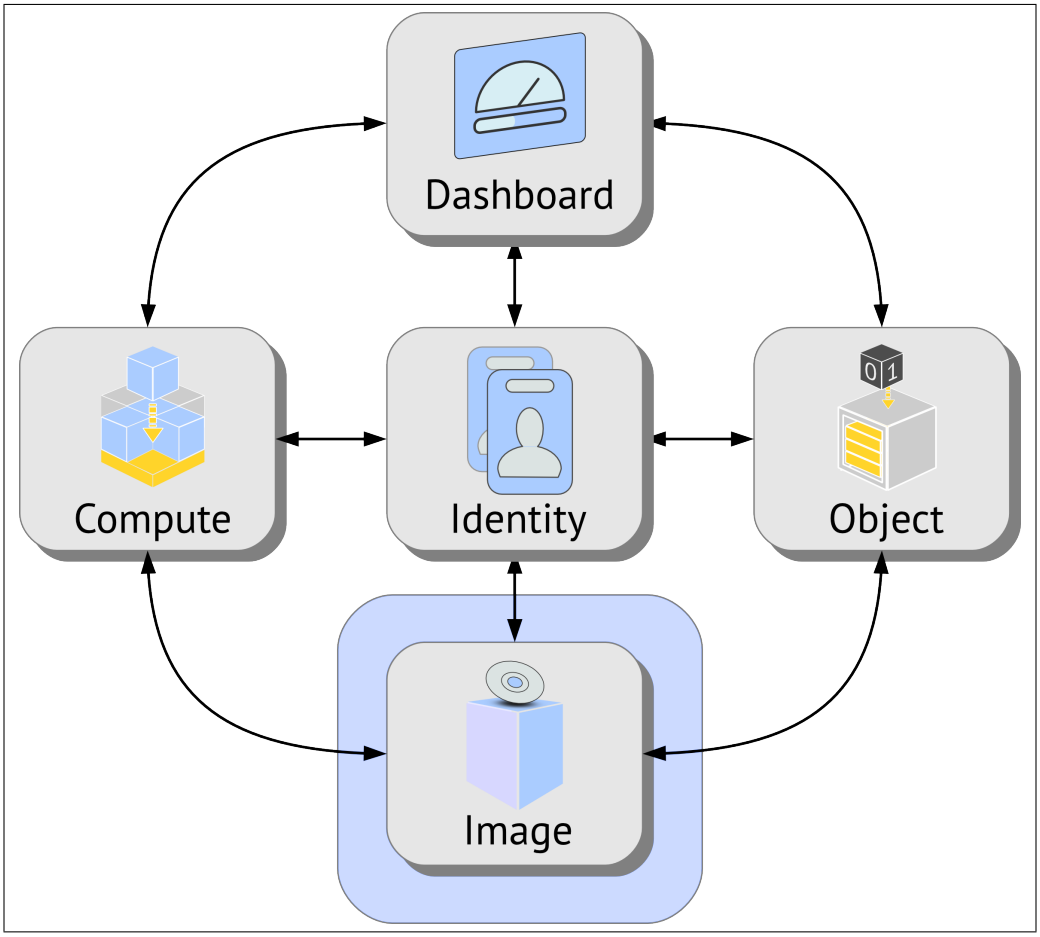

Abbildung 5: Beziehung anderer Komponenten zu *Glance*

Glance ist ein Dienst zum Entdecken, Registrieren und Verwalten von Imagedateien. Dabei unterstützt Glance verschiedene Storagebackends, z.B. den OpenStack-eigenen Swift.

Im Folgenden muss auf die korrekte Verteilung der Pakete auf die unterschiedlichen Hosts geachtet werden, die API wird auf dem Host `glance-api` installiert, die Registry hingegen auf `glance-registry`.

Glance selbst speichert eigene Verwaltungsdaten in einer Datenbank, die Images selbst im entsprechenden Storage-Backend.

Glance unterstützt folgende Storage-Backends:

- Local Storage
- Swift

- S3
- Ceph (RBD)
- HTTP

In dieser Dokumentation werden wir das lokale Dateisystem für die Ablage der Images nutzen. Wenn wir Swift aufnehmen, werden wir an dieser Stelle das Zusammenspiel von Glance und Swift erläutern.

Architektur

API (glance-api)

Auf dem Host `glance-api.openstack.b1-systems.de` wird die API von Glance installiert.

Installation

Um die `glance-api` zu installieren, wird das entsprechende Paket mit Hilfe von `zypper` installiert:

```
# zypper install openstack-glance-api
```

Die Konfiguration von Glance befindet sich in Dateien unterhalb von `/etc/glance`.

Konfiguration

```
[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = keystone.openstack.b1-systems.de
service_port = 5000
auth_host = keystone.openstack.b1-systems.de
auth_port = 35357
auth_protocol = http
auth_uri = http://keystone.openstack.b1-systems.de:5000/
admin_token = testing
```

In der Datei `/etc/glance/glance-api.conf` muss der Wert für das `rabbitmq`-Passwort angepasst werden, sowie der Host mit der Glance-Registry definiert werden:

```
registry_host = glance-registry.openstack.b1-systems.de
```

Die Zeilen:

```
[paste_deploy]
flavor = keystone
```

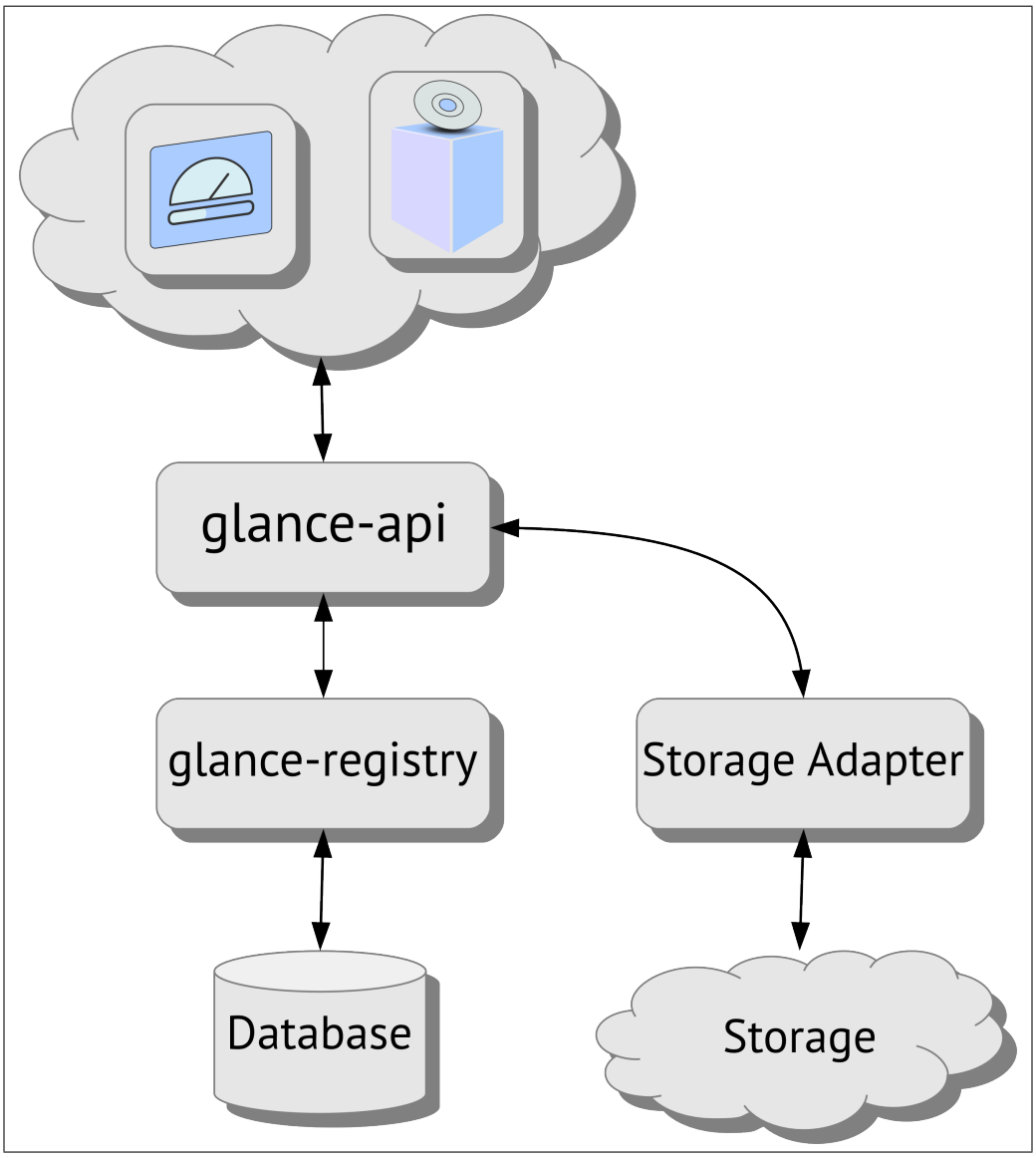

Abbildung 6: *Glance* besteht aus den Diensten `glance-api` und `glance-registry`

müssen an das Ende der Datei hinzugefügt werden.

Start

Auch hier muss wieder dafür gesorgt werden, dass der Dienst `openstack-glance-api` beim Start des Server initialisiert wird:

```
# insserv openstack-glance-api
# service openstack-glance-api start
```

Mit Hilfe von `netstat` kann der Zustand geprüft werden:

```
# netstat -tulpen | grep python
tcp        0      0 0.0.0.0:9292            0.0.0.0:*
   LISTEN      0          11807      1939/python
```

Registry (glance-registry)

Auf dem Host `glance-registry.openstack.b1-systems.de` wird die Registry von Glance installiert. Auch diese nutzt wieder Konfigurationsdateien unterhalb von `/etc/glance`.

Installation

Um die `glance-registry` zu installieren, wird das entsprechende Paket mit Hilfe von `zypper` installiert:

```
# zypper install openstack-glance-registry
```

Konfiguration

Die zentralen Konfigurationsdateien von `glance-registry` sind `glance-registry-paste.ini` und `glance-registry.conf` (beide zu finden unter `/etc/glance/`). Bei der Anpassung der `/etc/glance/glance-api-paste.ini` muss, wie schon bei der API, auf den Abschnitt `filter:authtoken` geachtet werden:

```
[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = keystone.openstack.b1-systems.de
service_port = 5000
auth_host = keystone.openstack.b1-systems.de
auth_port = 35357
auth_protocol = http
auth_uri = http://keystone.openstack.b1-systems.de:5000/
admin_token = testing
```

Die `/etc/glance/glance-registry.conf` hingegen bekommt die für die Datenbank notwendigen Werte:

```
connection = mysql://glance:testing@database.openstack.b1-systems.de/
   glance
```

Wie vorab bei der API erwähnt, die beiden folgenden Zeilen sind nicht in der ausgelieferten Konfigurationsdatei enthalten und müssen eingetragen werden:

```
[paste_deploy]
flavor = keystone
```

Nach den erfolgten Anpassungen muss die Datenbank initialisiert werden, dies geschieht per `glance-manage`:

```
# glance-manage db_sync
```

Start

Auch dieser Dienst wird für den automatischen Start beim Hochfahren des Servers eingestellt und anschließend gestartet:

```
# insserv openstack-glance-registry
# service openstack-glance-registry start
```

```
# netstat -tulpen | grep python
tcp        0      0 0.0.0.0:9191            0.0.0.0:*
    LISTEN      0          11548      1885/python
```

Benutzung

Damit Compute Images für die einzelnen VMs kopieren und zur Verfügung stellen kann, müssen diese in Glance registriert werden. Dabei wird das entsprechende Image in das konfigurierte Storage-Backend hochgeladen. Ab diesem Zeitpunkt kann es genutzt werden. Startet Compute nun eine virtuelle Maschine, wird verglichen, ob ein entsprechendes Image bereits im Template-Verzeichnis vorhanden ist. Ist das der Fall, wird das dort vorliegende Image mit Hilfe einer Prüfsumme mit dem in *Glance* hinterlegten verglichen; unterscheiden sich die Ergebnisse, wird das Image neu von *Glance* angefordert und heruntergeladen.
Durch diese Kontrolle wird unnötigtes Kopieren der Images vermieden.

Hinzufügen eines Images

Zur Bereitstellung eines kleinen Images in *Glance* wird ein vorhandenes Image aus dem Internet heruntergeladen, entpackt und in *Glance* registriert. Das Image selbst wird per `wget` gezogen und entpackt:

```
# wget http://images.ansolabs.com/tty.tgz
# tar xvzf tty.tgz
```

Danach stehen drei verschiedene Dateitypen zur Verfügung, eine Datei für die Ramdisk, eine für den Kernel und die letzte für das eigentliche Diskimage. Jede einzelne

muss nun an *Glance* registriert werden. Dazu wird das im Abschnitt *Keystone* erzeugte `TOKEN` genutzt. Nach der Registrierung eines Images wird eine UUID zurückgegeben, diese ist im letzten Beispiel notwendig, um den entsprechenden Kernel und die entsprechende Ramdisk mit dem Diskimage zu verknüpfen. Hier muss also angepasst werden:

```
# glance -A $TOKEN add name="tty-kernel" is_public=true
   container_format=aki disk_format=aki < images/aki-tty/image
# glance -A $TOKEN add name="tty-ramdisk" is_public=true
   container_format=ari disk_format=ari < images/ari-tty/image
# glance -A $TOKEN add name="tty" is_public=true container_format=ami
    disk_format=ami kernel_id=81c88115-a5b0-4421-9576-18a303c15ef0
   ramdisk_id=888c4a42-b2e0-4dbf-97c8-8d79b1db385c < images/ami-tty/
   image
```

Anzeigen verfügbarer Images

Zur Kontrolle, welche Images hinterlegt sind, wird der Parameter `index` von *Glance* genutzt:

```
# glance -A $TOKEN index
ID                                   Name
   Disk Format          Container Format     Size
------------------------------------ ------------------------------
   -------------------- -------------------- --------------
cba65750-10c0-4bab-83d1-9c5455d205ba tty
   ami                  ami                        25165824
4c23fe82-20f3-41d5-a845-8bd24df07037 tty-ramdisk
   ari                  ari                         5882349
ce95c812-02b2-4697-b819-58cbc8e76004 tty-kernel
   aki                  aki                         4404752
```

Löschen eines Images

Muss ein fehlerhaftes oder nicht mehr genutztes Image aus *Glance* entfernt werden, geschieht dies unter Angabe der UUID des Images. Hier ein Beispiel:

```
# glance -A $TOKEN delete ce95c812-02b2-4697-b819-58cbc8e76004
```

Nova

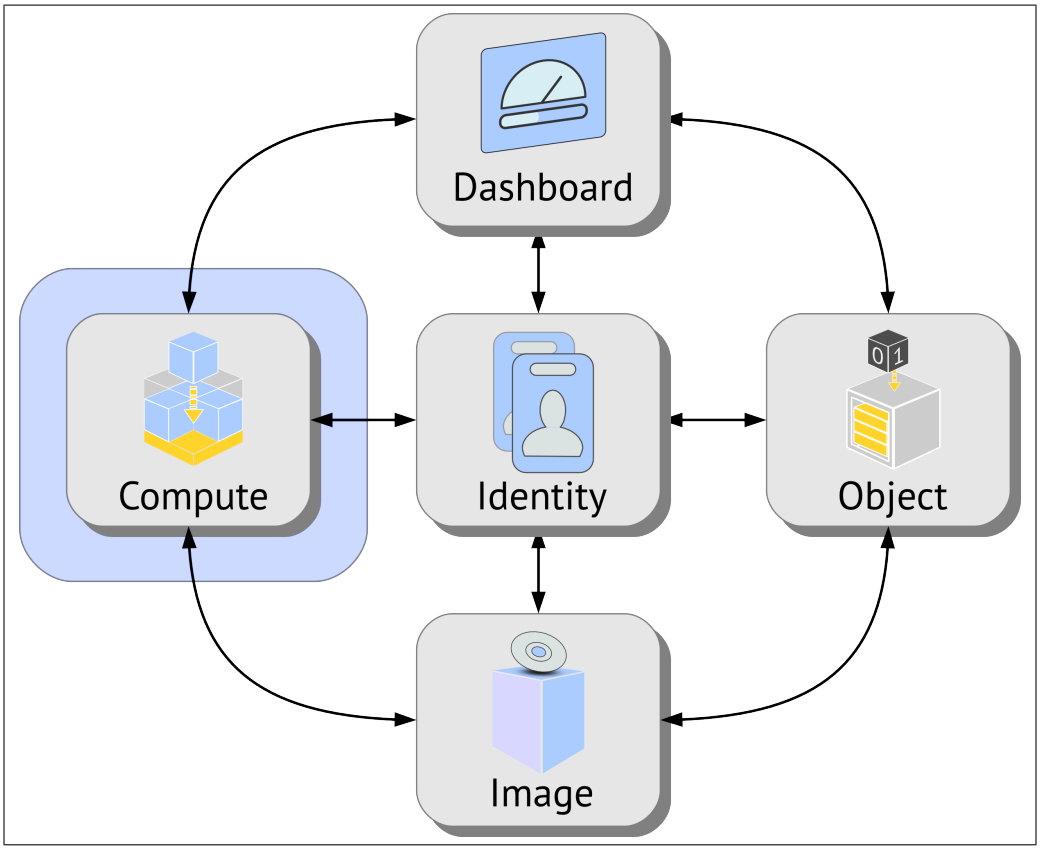

Abbildung 7: Zusammenhang der Komponenten

Die Komponente *OpenStack Compute* oder kurz *nova* ist für die Verwaltung virtueller Maschinen zuständig und besteht aus mehreren einzelnen Diensten. Es ist möglich, eine OpenStack-Installation jederzeit um einen weiteren nova-Host durch die Installation des entsprechenden Dienstes auf einem Server zu erweitern. Im Folgenden werden kurz die notwendigen Schritte zur Installation von *nova* demonstriert.

Bitte beachten Sie vorab die im Abschnitt *Anmerkungen* auf Seite 13 enthaltenen Informationen!

OpenStack Compute hat die Möglichkeit, mit unterschiedlichen Hypervisoren zu arbeiten. Dazu zählen u.a. Xen und KVM. Dies wird über die Zuweisung von `libvirt_type` in der Konfigurationsdatei `/etc/nova/nova.conf` erledigt. Das Projekt selbst verwendet KVM, zu Demonstrationszwecken wird in diesem Dokument Xen genutzt.

nova-api

Mit *nova* wird über dessen API kommuniziert. Dieser Dienst muss auf einem Host installiert werden; hier ist es das System mit dem Namen `nova-api.openstack.b1-systems.de`. Die Installation erfolgt durch einen Aufruf von `zypper`:

```
# zypper install openstack-nova-api
```

Die Konfiguration der API findet in der Datei `/etc/nova/api-paste.ini` statt. Beim Bearbeiten muss hier auch wieder auf den Abschnitt `filter:authtoken` geachtet werden:

```
[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = keystone.openstack.b1-systems.de
service_port = 5000
auth_host = keystone.openstack.b1-systems.de
auth_port = 35357
auth_protocol = http
auth_uri = http://keystone.openstack.b1-systems.de:5000/
admin_tenant_name = b1systems
admin_user = admin
admin_password = testing
```

Zusätzlich müssen in der `/etc/nova/nova.conf` noch globale Einstellungen konfiguriert werden, z.B. die Anbindung der Datenbank. Dazu kopiert man die mitgelieferte `/etc/nova/nova.conf.sample` nach `/etc/nova/nova.conf` oder trägt lediglich relevante Teile dort ein:

```
[DEFAULT]
verbose=true
debug=true
api_paste_config=/etc/nova/api-paste.ini
osapi_compute_extension=nova.api.openstack.compute.contrib.
    standard_extensions
sql_connection=mysql://nova:testing@database.openstack.b1-systems.de/
    nova
rabbit_host=rabbitmq.openstack.b1-systems.de
rabbit_password=testing
image_service=nova.image.glance.GlanceImageService
glance_api_servers=glance-api.openstack.b1-systems.de:9292
auth_strategy=keystone
```

Aktuell müssen noch eventuell gewährte Rechte in der Datei `/etc/nova/policy.json` eingetragen werden. Allerdings sollte hier die vom Paket mitgelieferte Datei ausreichen, so dass für den normalen Betrieb keine Änderungen gemacht werden müssen.

Wenn die API beim Start des Server initialisiert werden soll, so sorgen folgende Befehle dafür, dass dass der Dienst zuerst aktiviert und danach gestartet wird:

```
# chkconfig openstack-nova-api on
# service openstack-nova-api start
```

Nach der initialen Konfiguration kann das Schema in der Datenbank `nova` installiert werden. Dies erledigt ein Aufruf von:

```
# nova-manage db sync
```

nova-scheduler

Der Scheduler ist für das Planen von Aufträgen zuständig, er wird über `/etc/nova/nova.conf` konfiguriert.

Das entsprechende Paket wird auf dem Host `nova-scheduler` installiert:

```
# zypper install openstack-nova-scheduler
```

Um den einfachen Scheduler zu nutzen, benötigt man keine Änderung an der Konfigurationsdatei, da der Eintrag

```
scheduler_driver=nova.scheduler.simple.SimpleScheduler
```

bereits im Standard definiert ist.

nova-network

Dieser Teil ist aktuell noch für die Konfiguration des Netzwerks zuständig. Das bedeutet, dass `nova-network` auf dem entsprechenden Host Regeln für den Paketfilter erzeugt und entfernt.

Hierzu wird das benötigte Paket installiert

```
# zypper install openstack-nova-network
```

und natürlich für den automatischen Start aktiviert:

```
# chkconfig openstack-nova-network on
```

Gestartet wird der Dienst nun einmalig mit:

```
# service openstack-nova-network start
```

nova-volume

`nova-volume` bietet die Möglichkeit, beständige Imagedateien anzulegen, die exklusiv in einer laufenden VM genutzt werden können. Die auf dieses Volume gespeicherten Daten bleiben nach Beenden der VM erhalten, alle anderen Imagedateien werden beim Terminieren der Instanz entfernt. `nova-volume` übernimmt die Bereitstellung solcher beständiger Imagedateien.

Für die Nutzung von `nova-volume` muss eine frei wählbare VolumeGroup (LVM2) existieren. Der Standardname lautet `nova-volumes`, allerdings kann der Name der VolumeGroup über den Parameter

```
volume_group=openstack
```

in `/etc/nova/nova.conf` verändert werden, in diesem Fall auf `openstack`.

Für den Betrieb werden die benötigten Pakete installiert:

```
# zypper install openstack-nova-volume iscsitarget
```

Dann werden beide Prozesse gestartet:

```
# chkconfig iscsitarget on
# chkconfig openstack-nova-volume on
# rciscsitarget start
# /etc/init.d/openstack-nova-volume start
```

Nun besteht die Möglichkeit, Volumes anzulegen und in Instanzen zu verwenden. Dies geschieht am einfachsten über das Webinterface.

nova-compute

Auch `nova-compute` muss auf jedem Host, der virtuelle Maschinen starten soll, installiert werden. Dies wird per Aufruf von

```
# zypper install openstack-nova-compute
```

auf allen entsprechenden Hosts erledigt (nova-compute-002 und os0012).

In der zentralen Konfigurationsdatei von *nova* (`/etc/nova/nova.conf` werden einige Einträge für `compute` ergänzt:

```
instances_path=/var/lib/nova/instances
connection_type=libvirt
libvirt_type=xen
instance_name_template=instance-%08x
```

Achten Sie auf den Wert des `instances_path`, sollte das Verzeichnis nicht existieren, so erstellen Sie es per `mkdir -p`.

Auch dieser Dienst muss wieder für den automatischen Start aktiviert und gestartet werden:

```
# insserv openstack-nova-compute
# service openstack-nova-compute start
```

Prüfung auf `nova-api` (`nova-api`):

```
nova-api:~ # nova-manage service list
Binary           Host                                 Zone
                Status     State Updated_At
nova-scheduler   nova-scheduler                                nova
                enabled    :-)   2012-03-27 13:37:52
nova-compute     clementine                           nova
                enabled    :-)   2012-03-27 13:37:52
```

Administration

Images anzeigen

Zur Kontrolle ob *nova* die in Glance verfügbaren Images sieht, wird per `nova` eine Imageliste abgefragt. Im Hintergrund leitet `nova` diese Anfrage an Glance weiter. Hiermit kann also sofort die Funktionstüchtigkeit der Anbindung geprüft werden:

```
 # nova --os_username admin --os_password testing --os_tenant_name
    b1systems --os_auth_url http://keystone.openstack.b1-systems.de
    :5000/v2.0 image-list
+--------------------------------------+-------------+--------+--------+
|                  ID                  |     Name    | Status |
   Server |
+--------------------------------------+-------------+--------+--------+
| 4c23fe82-20f3-41d5-a845-8bd24df07037 | tty-ramdisk | ACTIVE |
          |
| cba65750-10c0-4bab-83d1-9c5455d205ba | tty         | ACTIVE |
          |
| ce95c812-02b2-4697-b819-58cbc8e76004 | tty-kernel  | ACTIVE |
          |
+--------------------------------------+-------------+--------+--------+
```

Flavors

OpenStack Compute stellt unterschiedliche Konfigurationen von virtuellen Maschinen mit sogenannten *Flavors* zur Verfügung. Dabei unterscheiden sich diese in der Anzahl der vCPUs, des Arbeitsspeichers und des zur Verfügung stehenden Plattenplatzes. Diese Flavors können abgefragt werden:

```
# nova --os_username admin --os_password testing --os_tenant_name
   b1systems --os_auth_url http://keystone.openstack.b1-systems.de
   :5000/v2.0 flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+
| ID |    Name   | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
   RXTX_Factor |
+----+-----------+-----------+------+-----------+------+-------+-------------+
| 1  | m1.tiny   | 512       | 0    | 0         |      | 1     | 1.0
           |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
           |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
           |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
           |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
           |
+----+-----------+-----------+------+-----------+------+-------+-------------+
```

Hinterlegen eines SSH-Keys

Einige Images unterstützen, dass vor dem Start der VM ein SSH-Key injiziert wird. Dabei wird das für den Start der VM vorbereitete Image einmalig gemountet und der entsprechende SSH-Key an die korrekte Stelle des Mountpunkts kopiert. Damit ist es nach dem Start der VM möglich, sich passwortlos per SSH auf die VM zu verbinden. Dazu muss dem Benutzer ein entsprechender Schlüssel zugewiesen werden:

```
# nova --os_username admin --os_password testing --os_tenant_name
   b1systems --os_auth_url http://keystone.openstack.b1-systems.de
   :5000/v2.0 keypair-add --pub_key ~/.ssh/id_rsa.pub mykey
```

Sollte bisher kein SSH Key zur Verfügung stehen, kann dieser über das Kommando `ssh-keygen` mit entsprechender Optionen erzeugt werden.

Starten einer VM

Damit eine virtuelle Maschine gestartet werden kann, muss eine entsprechende ID eines `flavors` bekannt sein, sowie die UUID des gewünschten Images. Zusätzlich kann noch ein in die VM zu kopierender SSH-Key angegeben werden. Ein Name ist notwendig. Der Aufbau des Befehls ist folgendermaßen:

```
nova boot --flavor ID --image ID --key_name KEY_NAME Name
```

Im folgenden Beispiel wird eine entsprechende VM gestartet:

```
# nova --os_username admin --os_password testing --os_tenant_name
   b1systems --os_auth_url http://keystone.openstack.b1-systems.de
   :5000/v2.0 boot --flavor 1 --image 0d992e04-9ee7-43d7-9312-93
   d21ad39240 --key_name mykey test
```

Anzeige aktiver VMs

Mit Hilfe des Parameters `list` werden aktuelle virtuelle Maschinen ausgegeben, inklusive des Status:

```
# nova --os_username admin --os_password testing --os_tenant_name
   b1systems --os_auth_url http://keystone.openstack.b1-systems.de
   :5000/v2.0 list
```

Beenden einer VM

Eine laufende virtuelle Maschine wird per `nova` beendet, dazu wird die UUID der gewünschten VM benötigt:

```
# nova --os_username admin --os_password testing --os_tenant_name
   b1systems --os_auth_url http://keystone.openstack.b1-systems.de
   :5000/v2.0 delete a45fd6be-f816-47fb-a789-6b2c0d6da55a
```

VNC-Console

Es besteht die Möglichkeit, sich per VNC die Konsole der Instanz zu holen und so zu arbeiten. Dazu werden zwei Pakete installiert:

```
# zypper in openstack-nova-novncproxy
```

Die beiden neuen Dienste werden so konfiguriert, dass sie beim Start des Servers initialisiert werden:

```
# chkconfig openstack-nova-consoleauth on
# chkconfig openstack-nova-novncproxy on
```

Einmalig werden diese Dienste manuell gestartet:

```
# /etc/init.d/openstack-nova-consoleauth start
# /etc/init.d/openstack-nova-novncproxy start
```

Der zusätzliche Parameter in der `/etc/nova/nova.conf` lautet:

```
novncproxy_base_url=http://nova-api.openstack.b1-systems.de:6080/
    vnc_auto.html
```

Nun sollte der Zugriff per VNC über das Dashboard funktionieren.

Horizon

Horizon stellt akutell ein Webinterface für OpenStack Compute bereit. In der Zukunft soll ein weites Funktionsspektrum aller Core-Komponenten nutzbar sein. *Horizon* nutzt das in Python geschriebene Webframework *Django*.

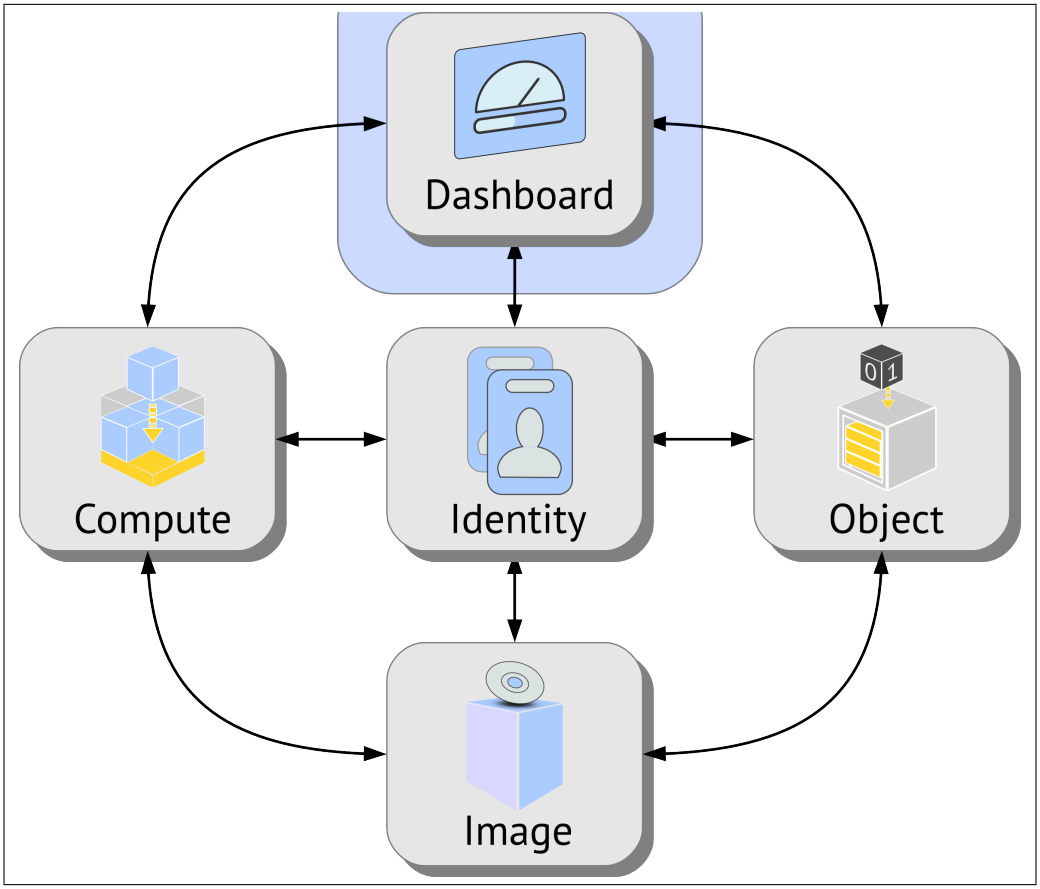

Abbildung 8: Beziehung anderer Komponenten zu *Horizon*

Architektur

Als Basis für Horizon dient ein Apache2-Webserver inklusive `mod_wsgi`, einer Schnittstelle zum Ausführen entsprechender Python-Applikationen. Dementsprechend muss das Modul geladen und der Webserver korrekt konfiguriert sein.
Horizon selbst kann Benutzer gegen Keystone authentifizieren. Somit wird eine einfache Anwendung möglich.

Installation

Zunächste werden `apache2`, `apache2-mod_wsgi`, `memcached` und `openstack-dashboard` installiert:

```
# zypper install memcached openstack-dashboard apache2 apache2-
   mod_wsgi python-memcache
```

Nach der Installation wird `memached` für den automatischen Start konfiguriert und einmalig manuell gestartet. Wir behalten die Default-Konfiguration von `memcached` bei.

```
# chkconfig memcached on
# service memcached start
```

Konfiguration

Im Paket `openstack-dashboard` ist eine funktionierende Beispielkonfiguration für Apache2 enthalten. Diese wird in das Konfigurationsverzeichnis von Apache2 kopiert:

```
# cp /etc/dashboard/dashboard.conf /etc/apache2/conf.d/
```

Die Konfiguration erstellt einen neuen WSGI-Prozess für Django und stellt diese unter dem Pfad `/` zur Verfügung:

```
WSGIDaemonProcess dashboard
WSGIScriptAlias / /usr/share/openstack_dashboard/wsgi/django.wsgi

<Directory /usr/share/openstack_dashboard/wsgi/>
  Order allow,deny
  Allow from all
</Directory>
```

Nun wird die Konfiguration für Dashboard wie folgt erstellt:

```
# cd /usr/share/openstack_dashboard/local
# cp local_settings.py.example local_settings.py
```

Zur Verwendung von `memcached` wird in dieser Datei der Parameter `CACHE_BACKEND` wie folgt angepasst:

```
CACHE_BACKEND = 'memcached://127.0.0.1:11211/'
```

Das Dashboard verwendet wie alle anderen Komponenten den zentralen Identifizierungsdienst Keystone zur Validierung von Logins sowie zum Bezug von Projekt- und Benutzerdetails. Die Anbindung an Keystone erfolgt entsprechend in der Konfiguration:

```
OPENSTACK_HOST = "keystone.openstack.b1-systems.de"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"
```

Der Parameter `SECRET_KEY` sollte auf einen zufälligen Wert geändert werden, z.B. die Checksumme einer UUID:

```
# uuidgen | sha1sum
6d2320bb793f9e0fa6d4af0a872b790cfc401a64  -
```

```
SECRET_KEY = '6d2320bb793f9e0fa6d4af0a872b790cfc401a64'
```

Weiterhin wird die neue Rolle `Member` auf Keystone zugefügt. Benutzer, denen die Rolle `Member` zugeordnet werden, können den Kundenbereich des Dashboards verwenden. Benutzer, denen der Rollte `admin` zugewiesen sind, wird zusätzlich Zugriff auf den Adminbereich gewährt.

Start

Abschließend wird das Modul `wsgi` aktiviert und Apache2 gestartet:

```
# a2enmod wsgi
# insserv apache2
# service apache2 start
```

Vorbereitungen

Als Beispiel erstellen wir ein neues Projekt `horizon` mit den Benutzern `horizon_admin` und `horizon_user`

Hinweis: Die folgenden Parameter sind dem Aufruf von `keystone` jeweils zuzufügen. Wir führen diese aus Platzgründen nicht auf.
`--endpoint http://keystone.openstack.b1-systems.de:35357/v2.0/ --token testing`.

Alternativ können diese Werte auch über die Umgebungsvariablen `SERVICE_ENDPOINT` sowie `SERVICE_TOKEN` der CLI bekannt gemacht werden.

```
export SERVICE_ENDPONT="http://keystone.openstack.b1-systems.de
   :35357/v2.0/"
export SERVICE_TOKE="testing"
```

```
# keystone  testing role-create --name Member
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
```

```
| id       | 345fe8051d094b52b7190c6f5e83008c |
| name     | Member                           |
+----------+----------------------------------+
```

```
# keystone tenant-create --name horizon
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | None                             |
| enabled     | True                             |
| id          | dd8034a055e24bc2bfba03159759a3e1 |
| name        | horizon                          |
+-------------+----------------------------------+
```

```
# keystone user-create --name horizon_member --pass testing --
   tenant_id dd8034a055e24bc2bfba03159759a3e1
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
| email    | None                             |
| enabled  | True                             |
| id       | fba3956efd8040fe92298958f23b39aa |
| name     | horizon_member                   |
| password | $6$rounds=40000$PLQfnwap8nsnf... |
| tenantId | dd8034a055e24bc2bfba03159759a3e1 |
+----------+----------------------------------+
# keystone user-create --name horizon_admin --pass testing --
   tenant_id dd8034a055e24bc2bfba03159759a3e1
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
| email    | None                             |
| enabled  | True                             |
| id       | 2c0924e8f88f4ee5b193e366a60c81c5 |
| name     | horizon_admin                    |
| password | $6$rounds=40000$vzILEun2eN80/... |
| tenantId | dd8034a055e24bc2bfba03159759a3e1 |
+----------+----------------------------------+
```

Die zuvor erstellte Rolle `Member` kann nun bestehenden Nutzern zugefügt werden:

```
# keystone user-role-add --role $ID_OF_ROLE_MEMBER --user fba39... --
   tenant_id dd803...
# keystone user-role-add --role $ID_OF_ROLE_ADMIN --user ...2c092 --
   tenant_id dd803...
```

Das Dashboard ist jetzt unter der Adresse des Hosts über einen Browser erreichbar:

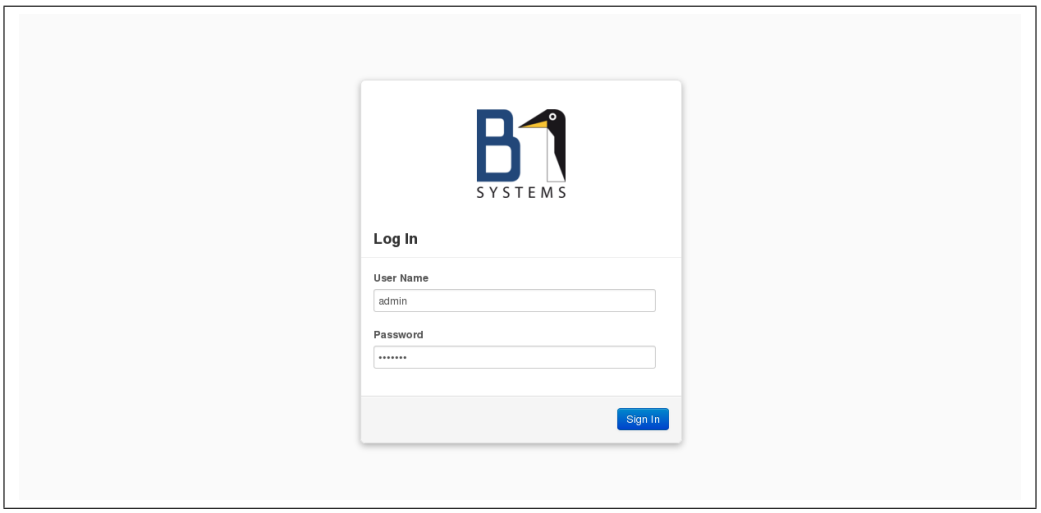

Abbildung 9: Login ist mit Benutzer `admin` und Passwort `testing` möglich

Benutzung

Beispiele aus dem Adminbereich

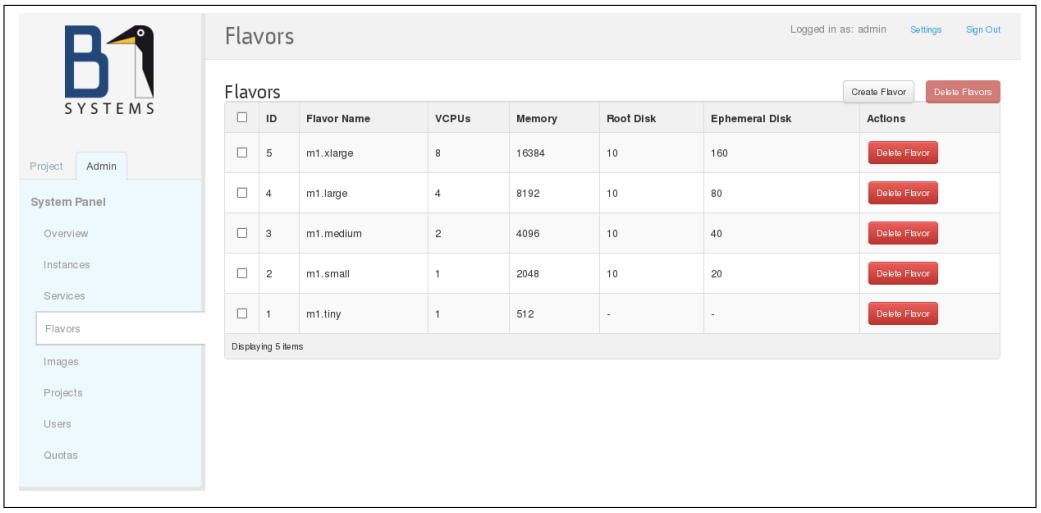

Abbildung 10: Verwaltung der sogenannten Flavors, der Typen von VMs

Beispiele aus dem Kundenbereich

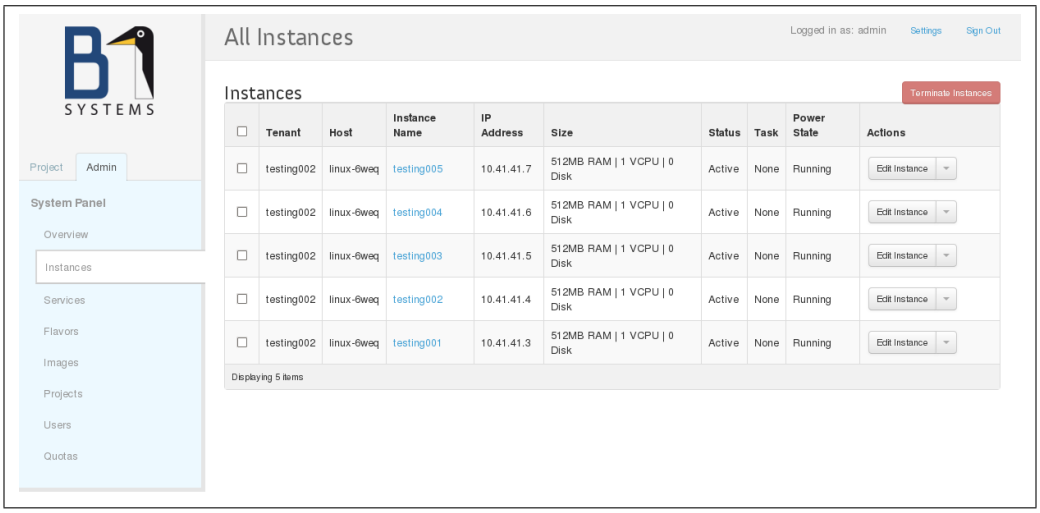

Abbildung 11: Übersicht aller Instanzen

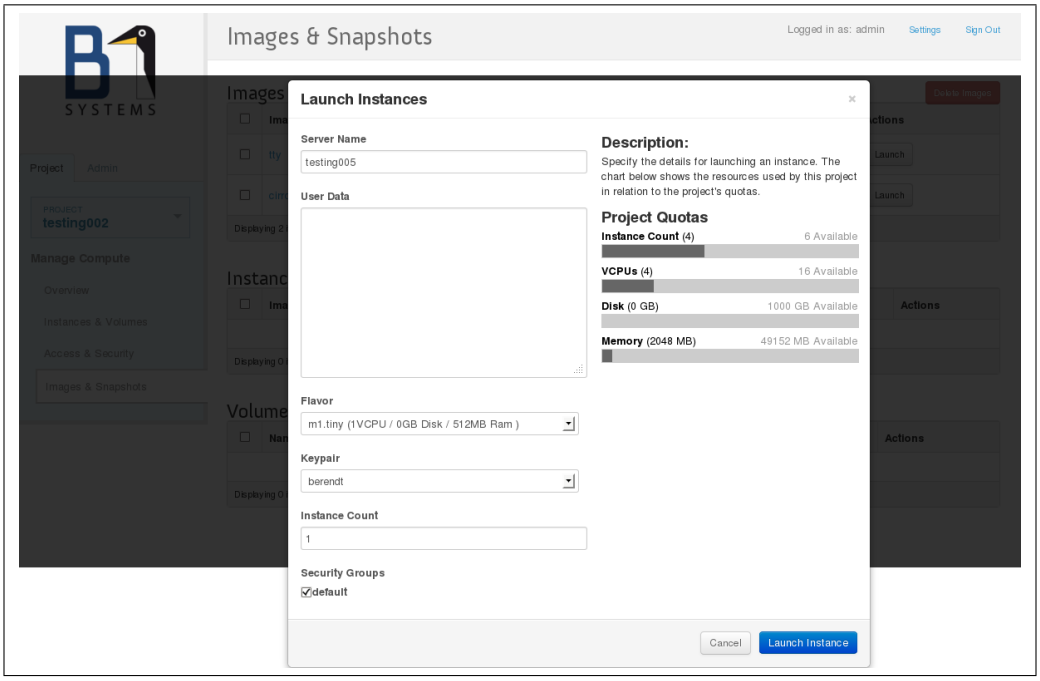

Abbildung 12: Erzeugen einer neuen Instanz

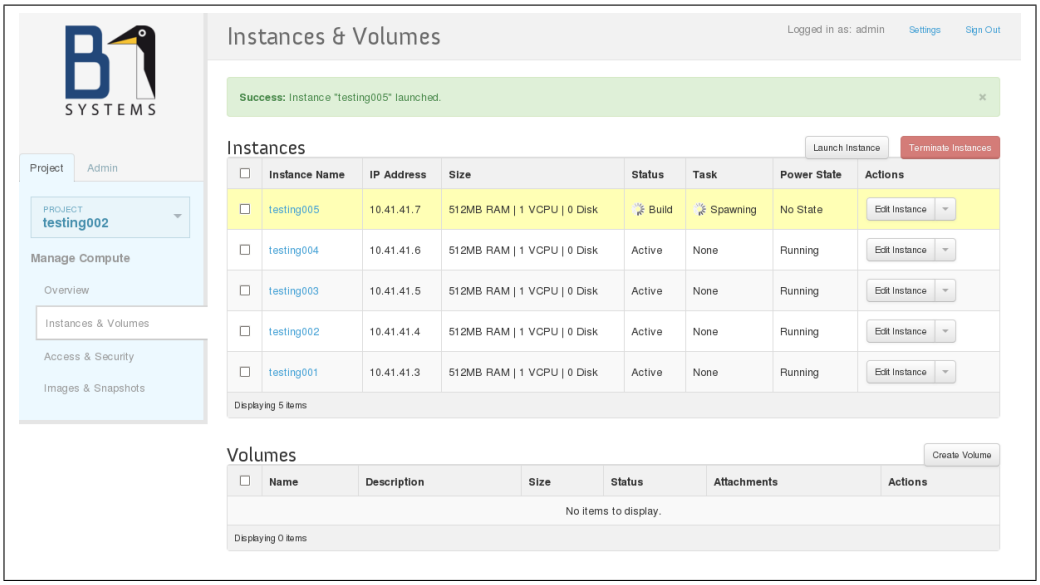

Abbildung 13: Übersicht der Instanzen und Volumes

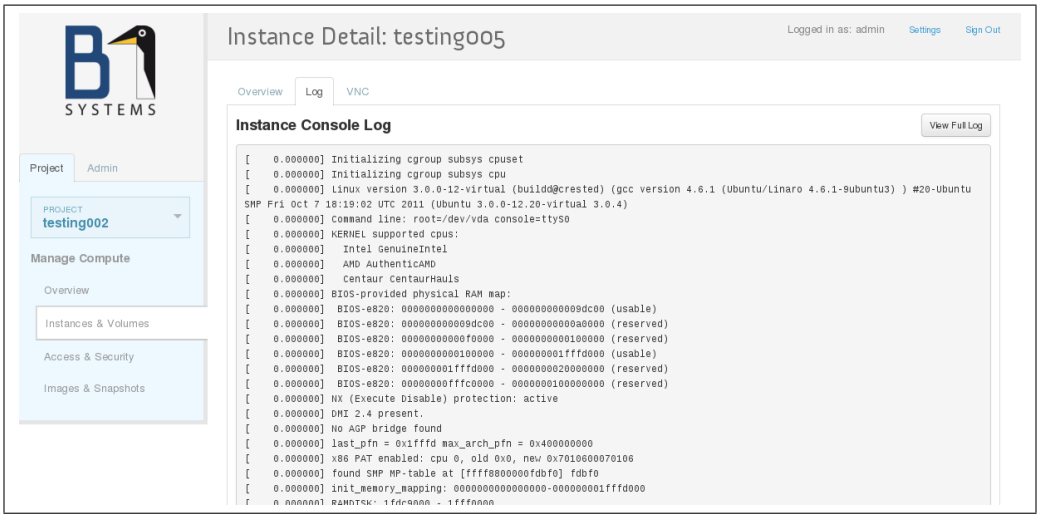

Abbildung 14: Das Log einer Instanz

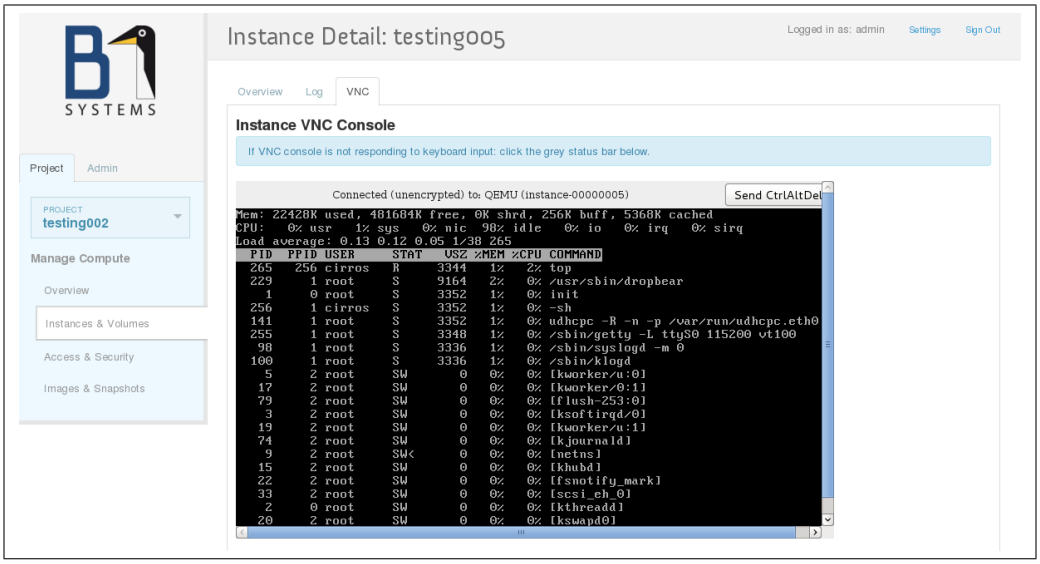

Abbildung 15: Zugriff per VNC auf eine Instanz

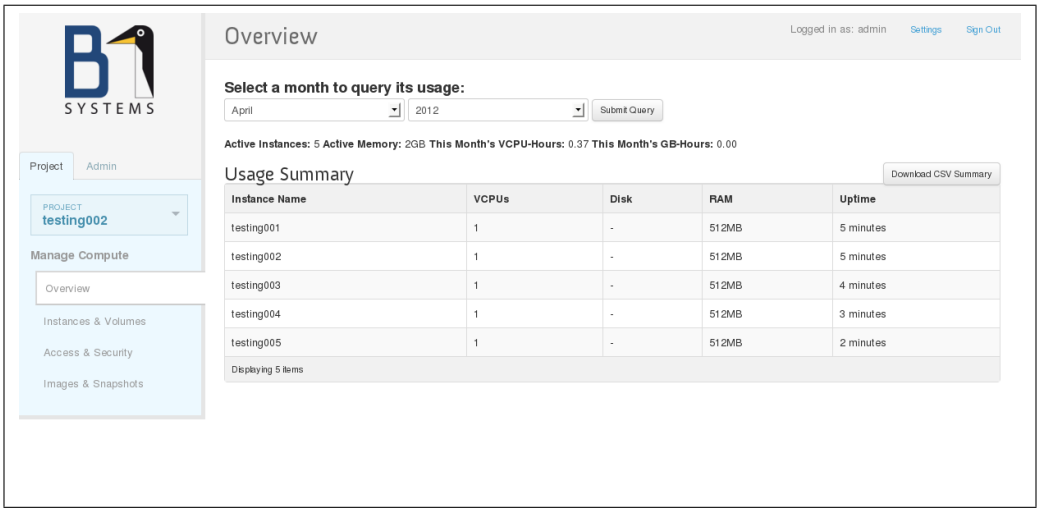

Abbildung 16: Nutzungsstatistik

Anhang

Keystone

/etc/keystone/default_catalog.templates

```
catalog.RegionOne.identity.publicURL = http://keystone.openstack.b1-
    systems.de:$(public_port)s/v2.0
catalog.RegionOne.identity.adminURL = http://keystone.openstack.b1-
    systems.de:$(admin_port)s/v2.0
catalog.RegionOne.identity.internalURL = http://keystone.openstack.b1
    -systems.de:$(admin_port)s/v2.0
catalog.RegionOne.identity.name = Identity Service

catalog.RegionOne.compute.publicURL = http://nova-api.openstack.b1-
    systems.de:$(compute_port)s/v1.1/$(tenant_id)s
catalog.RegionOne.compute.adminURL = http://nova-api.openstack.b1-
    systems.de:$(compute_port)s/v1.1/$(tenant_id)s
catalog.RegionOne.compute.internalURL = http://nova-api.openstack.b1-
    systems.de:$(compute_port)s/v1.1/$(tenant_id)s
catalog.RegionOne.compute.name = Compute Service

catalog.RegionOne.volume.publicURL = http://nova-api.openstack.b1-
    systems.de:8776/v1/$(tenant_id)s
catalog.RegionOne.volume.adminURL = http://nova-api.openstack.b1-
    systems.de:8776/v1/$(tenant_id)s
catalog.RegionOne.volume.internalURL = http://nova-api.openstack.b1-
    systems.de:8776/v1/$(tenant_id)s
catalog.RegionOne.volume.name = Volume Service

catalog.RegionOne.ec2.publicURL = http://nova-api.openstack.b1-
    systems.de:8773/services/Cloud
catalog.RegionOne.ec2.adminURL = http://nova-api.openstack.b1-systems
    .de:8773/services/Admin
catalog.RegionOne.ec2.internalURL = http://nova-api.openstack.b1-
    systems.de:8773/services/Cloud
catalog.RegionOne.ec2.name = EC2 Service

catalog.RegionOne.image.publicURL = http://glance-api.openstack.b1-
    systems.de:9292/v1
catalog.RegionOne.image.adminURL = http://glance-api.openstack.b1-
    systems.de:9292/v1
catalog.RegionOne.image.internalURL = http://glance-api.openstack.b1-
    systems.de:9292/v1
catalog.RegionOne.image.name = Image Service
```

/etc/keystone/keystone.conf

```
[DEFAULT]
bind_host = 0.0.0.0
public_port = 5000
admin_port = 35357
admin_token = testing
compute_port = 8774
verbose = True
debug = True
#log_config = /etc/keystone/logging.conf
log_file = /var/log/keystone/keystone.log

# ================ Syslog Options ============================
# Send logs to syslog (/dev/log) instead of to file specified
# by `log-file`
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL0

[sql]
connection = mysql://keystone:testing@database.openstack.b1-systems.
   de/keystone
idle_timeout = 60
min_pool_size = 5
max_pool_size = 10
pool_timeout = 200

[ldap]
#url = ldap://localhost
#tree_dn = dc=example,dc=com
#user_tree_dn = ou=Users,dc=example,dc=com
#role_tree_dn = ou=Roles,dc=example,dc=com
#tenant_tree_dn = ou=Groups,dc=example,dc=com
#user = dc=Manager,dc=example,dc=com
#password = freeipa4all
#suffix = cn=example,cn=com

[identity]
driver = keystone.identity.backends.sql.Identity

[catalog]
driver = keystone.catalog.backends.templated.TemplatedCatalog
template_file = /etc/keystone/default_catalog.templates

[token]
driver = keystone.token.backends.sql.Token

# Amount of time a token should remain valid (in seconds)
expiration = 86400
```

```
[policy]
driver = keystone.policy.backends.rules.Policy

[ec2]
driver = keystone.contrib.ec2.backends.sql.Ec2

[filter:debug]
paste.filter_factory = keystone.common.wsgi:Debug.factory

[filter:token_auth]
paste.filter_factory = keystone.middleware:TokenAuthMiddleware.
    factory

[filter:admin_token_auth]
paste.filter_factory = keystone.middleware:AdminTokenAuthMiddleware.
    factory

[filter:xml_body]
paste.filter_factory = keystone.middleware:XmlBodyMiddleware.factory

[filter:json_body]
paste.filter_factory = keystone.middleware:JsonBodyMiddleware.factory

[filter:crud_extension]
paste.filter_factory = keystone.contrib.admin_crud:CrudExtension.
    factory

[filter:ec2_extension]
paste.filter_factory = keystone.contrib.ec2:Ec2Extension.factory

[app:public_service]
paste.app_factory = keystone.service:public_app_factory

[app:admin_service]
paste.app_factory = keystone.service:admin_app_factory

[pipeline:public_api]
pipeline = token_auth admin_token_auth xml_body json_body debug
    ec2_extension public_service

[pipeline:admin_api]
pipeline = token_auth admin_token_auth xml_body json_body debug
    ec2_extension crud_extension admin_service

[app:public_version_service]
paste.app_factory = keystone.service:public_version_app_factory

[app:admin_version_service]
```

```
paste.app_factory = keystone.service:admin_version_app_factory

[pipeline:public_version_api]
pipeline = xml_body public_version_service

[pipeline:admin_version_api]
pipeline = xml_body admin_version_service

[composite:main]
use = egg:Paste#urlmap
/v2.0 = public_api
/ = public_version_api

[composite:admin]
use = egg:Paste#urlmap
/v2.0 = admin_api
/ = admin_version_api
```

/etc/keystone/logging.conf

```
[loggers]
keys=root

[formatters]
keys=normal,normal_with_name,debug

[handlers]
keys=production,file,devel

[logger_root]
level=WARNING
handlers=file

[handler_production]
class=handlers.SysLogHandler
level=ERROR
formatter=normal_with_name
args=(('localhost', handlers.SYSLOG_UDP_PORT), handlers.SysLogHandler
    .LOG_USER)

[handler_file]
class=FileHandler
level=DEBUG
formatter=normal_with_name
args=('keystone.log', 'a')

[handler_devel]
class=StreamHandler
```

```
level=NOTSET
formatter=debug
args=(sys.stdout,)

[formatter_normal]
format=%(asctime)s %(levelname)s %(message)s

[formatter_normal_with_name]
format=(%(name)s): %(asctime)s %(levelname)s %(message)s

[formatter_debug]
format=(%(name)s): %(asctime)s %(levelname)s %(module)s %(funcName)s
    %(message)s
```

/etc/keystone/policy.json

```
{
    "admin_required": [["role:admin"], ["is_admin:1"]]
}
```

Glance

/etc/glance/policy.json

```
{
    "default": [],
    "manage_image_cache": [["role:admin"]]
}
```

/etc/glance/glance-api.conf

```
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = True

# Which backend store should Glance use by default is not specified
# in a request to add a new image to Glance? Default: 'file'
# Available choices are 'file', 'swift', and 's3'
default_store = file

# Address to bind the API server
```

```
bind_host = 0.0.0.0

# Port the bind the API server to
bind_port = 9292

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/api.log

# Backlog requests when creating socket
backlog = 4096

# Number of Glance API worker processes to start.
# On machines with more than one CPU increasing this value
# may improve performance (especially if using SSL with
# compression turned on). It is typically recommended to set
# this value to the number of CPUs present on your machine.
workers = 0

# ================= Syslog Options ============================

# Send logs to syslog (/dev/log) instead of to file specified
# by `log_file`
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL0

# ================= SSL Options ===============================

# Certificate file to use when starting API server securely
# cert_file = /path/to/certfile

# Private key file to use when starting API server securely
# key_file = /path/to/keyfile

# ================= Security Options ==========================

# AES key for encrypting store 'location' metadata, including
# -- if used -- Swift or S3 credentials
# Should be set to a random string of length 16, 24 or 32 bytes
# metadata_encryption_key = <16, 24 or 32 char registry metadata key>

# ============ Registry Options ===============================

# Address to find the registry server
registry_host = glance-registry.openstack.b1-systems.de

# Port the registry server is listening on
```

```
registry_port = 9191

# What protocol to use when connecting to the registry server?
# Set to https for secure HTTP communication
registry_client_protocol = http

# The path to the key file to use in SSL connections to the
# registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_KEY_FILE environ variable to a filepath of the key
    file
# registry_client_key_file = /path/to/key/file

# The path to the cert file to use in SSL connections to the
# registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_CERT_FILE environ variable to a filepath of the cert
    file
# registry_client_cert_file = /path/to/cert/file

# The path to the certifying authority cert file to use in SSL
    connections
# to the registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_CA_FILE environ variable to a filepath of the CA cert
     file
# registry_client_ca_file = /path/to/ca/file

# ============ Notification System Options =====================

# Notifications can be sent when images are create, updated or
    deleted.
# There are three methods of sending notifications, logging (via the
# log_file directive), rabbit (via a rabbitmq queue), qpid (via a
    Qpid
# message queue), or noop (no notifications sent, the default)
notifier_strategy = rabbit

# Configuration options if sending notifications via rabbitmq (these
    are
# the defaults)
rabbit_host = rabbitmq.openstack.b1-systems.de
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = guest
rabbit_password = testing
rabbit_virtual_host = /
rabbit_notification_exchange = glance
rabbit_notification_topic = glance_notifications

# Configuration options if sending notifications via Qpid (these are
# the defaults)
```

```
qpid_notification_exchange = glance
qpid_notification_topic = glance_notifications
qpid_host = localhost
qpid_port = 5672
qpid_username =
qpid_password =
qpid_sasl_mechanisms =
qpid_reconnect_timeout = 0
qpid_reconnect_limit = 0
qpid_reconnect_interval_min = 0
qpid_reconnect_interval_max = 0
qpid_reconnect_interval = 0
qpid_heartbeat = 5
# Set to 'ssl' to enable SSL
qpid_protocol = tcp
qpid_tcp_nodelay = True

# ============ Filesystem Store Options ========================

# Directory that the Filesystem backend store
# writes image data to
filesystem_store_datadir = /var/lib/glance/images/

# ============ Swift Store Options =============================

# Address where the Swift authentication service lives
# Valid schemes are 'http://' and 'https://'
# If no scheme specified,  default to 'https://'
swift_store_auth_address = 127.0.0.1:8080/v1.0/

# User to authenticate against the Swift authentication service
# If you use Swift authentication service, set it to 'account':'user'
# where 'account' is a Swift storage account and 'user'
# is a user in that account
swift_store_user = jdoe:jdoe

# Auth key for the user authenticating against the
# Swift authentication service
swift_store_key = a86850deb2742ec3cb41518e26aa2d89

# Container within the account that the account should use
# for storing images in Swift
swift_store_container = glance

# Do we create the container if it does not exist?
swift_store_create_container_on_put = False

# What size, in MB, should Glance start chunking image files
# and do a large object manifest in Swift? By default, this is
```

```
# the maximum object size in Swift, which is 5GB
swift_store_large_object_size = 5120

# When doing a large object manifest, what size, in MB, should
# Glance write chunks to Swift? This amount of data is written
# to a temporary disk buffer during the process of chunking
# the image file, and the default is 200MB
swift_store_large_object_chunk_size = 200

# Whether to use ServiceNET to communicate with the Swift storage
    servers.
# (If you aren't RACKSPACE, leave this False!)
#
# To use ServiceNET for authentication, prefix hostname of
# `swift_store_auth_address` with 'snet-'.
# Ex. https://example.com/v1.0/ -> https://snet-example.com/v1.0/
swift_enable_snet = False

# ============ S3 Store Options =============================

# Address where the S3 authentication service lives
# Valid schemes are 'http://' and 'https://'
# If no scheme specified,  default to 'http://'
s3_store_host = 127.0.0.1:8080/v1.0/

# User to authenticate against the S3 authentication service
s3_store_access_key = <20-char AWS access key>

# Auth key for the user authenticating against the
# S3 authentication service
s3_store_secret_key = <40-char AWS secret key>

# Container within the account that the account should use
# for storing images in S3. Note that S3 has a flat namespace,
# so you need a unique bucket name for your glance images. An
# easy way to do this is append your AWS access key to "glance".
# S3 buckets in AWS *must* be lowercased, so remember to lowercase
# your AWS access key if you use it in your bucket name below!
s3_store_bucket = <lowercased 20-char aws access key>glance

# Do we create the bucket if it does not exist?
s3_store_create_bucket_on_put = False

# When sending images to S3, the data will first be written to a
# temporary buffer on disk. By default the platform's temporary
    directory
# will be used. If required, an alternative directory can be
    specified here.
# s3_store_object_buffer_dir = /path/to/dir
```

```
# ============ RBD Store Options =============================

# Ceph configuration file path
# If using cephx authentication, this file should
# include a reference to the right keyring
# in a client.<USER> section
rbd_store_ceph_conf = /etc/ceph/ceph.conf

# RADOS user to authenticate as (only applicable if using cephx)
rbd_store_user = glance

# RADOS pool in which images are stored
rbd_store_pool = images

# Images will be chunked into objects of this size (in megabytes).
# For best performance, this should be a power of two
rbd_store_chunk_size = 8

# ============ Delayed Delete Options =============================

# Turn on/off delayed delete
delayed_delete = False

# Delayed delete time in seconds
scrub_time = 43200

# Directory that the scrubber will use to remind itself of what to
    delete
# Make sure this is also set in glance-scrubber.conf
scrubber_datadir = /var/lib/glance/scrubber

# =============== Image Cache Options =============================

# Base directory that the Image Cache uses
image_cache_dir = /var/lib/glance/image-cache/

[paste_deploy]
flavor = keystone
```

/etc/glance/glance-registry.conf

```
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = True
```

```
# Address to bind the registry server
bind_host = 0.0.0.0

# Port the bind the registry server to
bind_port = 9191

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/registry.log

# Backlog requests when creating socket
backlog = 4096

# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
# See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/
    connections.html#sqlalchemy.create_engine
sql_connection = mysql://glance:testing@database.openstack.b1-systems
    .de/glance


# Period in seconds after which SQLAlchemy should reestablish its
    connection
# to the database.
#
# MySQL uses a default `wait_timeout` of 8 hours, after which it will
     drop
# idle connections. This can result in 'MySQL Gone Away' exceptions.
    If you
# notice this, you can lower this value to ensure that SQLAlchemy
    reconnects
# before MySQL can drop the connection.
sql_idle_timeout = 3600

# Limit the api to return `param_limit_max` items in a call to a
    container. If
# a larger `limit` query param is provided, it will be reduced to
    this value.
api_limit_max = 1000

# If a `limit` query param is not provided in an api request, it will
# default to `limit_param_default`
limit_param_default = 25

# ================= Syslog Options ============================

# Send logs to syslog (/dev/log) instead of to file specified
# by `log_file`
```

```
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL1

# ================ SSL Options ==============================

# Certificate file to use when starting registry server securely
# cert_file = /path/to/certfile

# Private key file to use when starting registry server securely
# key_file = /path/to/keyfile

[paste_deploy]
flavor = keystone
```

/etc/glance/glance-api-paste.conf

```
# Default minimal pipeline
[pipeline:glance-api]
pipeline = versionnegotiation context apiv1app

# Use the following pipeline for keystone auth
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone
#
[pipeline:glance-api-keystone]
pipeline = versionnegotiation authtoken context apiv1app

# Use the following pipeline to enable transparent caching of image
   files
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = caching
#
[pipeline:glance-api-caching]
pipeline = versionnegotiation context cache apiv1app

# Use the following pipeline for keystone auth with caching
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone+caching
#
[pipeline:glance-api-keystone+caching]
pipeline = versionnegotiation authtoken context cache apiv1app

# Use the following pipeline to enable the Image Cache Management API
```

```
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = cachemanagement
#
[pipeline:glance-api-cachemanagement]
pipeline = versionnegotiation context cache cachemanage apiv1app

# Use the following pipeline for keystone auth with cache management
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone+cachemanagement
#
[pipeline:glance-api-keystone+cachemanagement]
pipeline = versionnegotiation authtoken context cache cachemanage
   apiv1app

[app:apiv1app]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.api.v1.router:API

[filter:versionnegotiation]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.api.middleware.version_negotiation:
   VersionNegotiationFilter

[filter:cache]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.api.middleware.cache:CacheFilter

[filter:cachemanage]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.api.middleware.cache_manage:
   CacheManageFilter

[filter:context]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.common.context:ContextMiddleware

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = keystone.openstack.b1-systems.de
service_port = 5000
auth_host = keystone.openstack.b1-systems.de
auth_port = 35357
auth_protocol = http
auth_uri = http://keystone.openstack.b1-systems.de:5000/
admin_tenant_name = b1systems
admin_user = admin
```

```
admin_password = testing
#admin_token = testing
```

/etc/glance/glance-api-registry.conf

```
# Default minimal pipeline
[pipeline:glance-registry]
pipeline = context registryapp

# Use the following pipeline for keystone auth
# i.e. in glance-registry.conf:
#   [paste_deploy]
#   flavor = keystone
#
[pipeline:glance-registry-keystone]
pipeline = authtoken context registryapp

[app:registryapp]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.registry.api.v1:API

[filter:context]
context_class = glance.registry.context.RequestContext
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.common.context:ContextMiddleware

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = keystone.openstack.b1-systems.de
service_port = 5000
auth_host = keystone.openstack.b1-systems.de
auth_port = 35357
auth_protocol = http
auth_uri = http://keystone.openstack.b1-systems.de:5000/
admin_tenant_name = b1systems
admin_user = admin
admin_password = testing
#admin_token = testing
```

Nova

/etc/nova/api-paste.ini

```
############
# Metadata #
```

```
############
[composite:metadata]
use = egg:Paste#urlmap
/: metaversions
/latest: meta
/1.0: meta
/2007-01-19: meta
/2007-03-01: meta
/2007-08-29: meta
/2007-10-10: meta
/2007-12-15: meta
/2008-02-01: meta
/2008-09-01: meta
/2009-04-04: meta

[pipeline:metaversions]
pipeline = ec2faultwrap logrequest metaverapp

[pipeline:meta]
pipeline = ec2faultwrap logrequest metaapp

[app:metaverapp]
paste.app_factory = nova.api.metadata.handler:Versions.factory

[app:metaapp]
paste.app_factory = nova.api.metadata.handler:MetadataRequestHandler.
    factory

#######
# EC2 #
#######

[composite:ec2]
use = egg:Paste#urlmap
/services/Cloud: ec2cloud

[composite:ec2cloud]
use = call:nova.api.auth:pipeline_factory
noauth = ec2faultwrap logrequest ec2noauth cloudrequest validator
    ec2executor
deprecated = ec2faultwrap logrequest authenticate cloudrequest
    validator ec2executor
keystone = ec2faultwrap logrequest ec2keystoneauth cloudrequest
    validator ec2executor

[filter:ec2faultwrap]
paste.filter_factory = nova.api.ec2:FaultWrapper.factory

[filter:logrequest]
```

```
paste.filter_factory = nova.api.ec2:RequestLogging.factory

[filter:ec2lockout]
paste.filter_factory = nova.api.ec2:Lockout.factory

[filter:totoken]
paste.filter_factory = nova.api.ec2:EC2Token.factory

[filter:ec2keystoneauth]
paste.filter_factory = nova.api.ec2:EC2KeystoneAuth.factory

[filter:ec2noauth]
paste.filter_factory = nova.api.ec2:NoAuth.factory

[filter:authenticate]
paste.filter_factory = nova.api.ec2:Authenticate.factory

[filter:cloudrequest]
controller = nova.api.ec2.cloud.CloudController
paste.filter_factory = nova.api.ec2:Requestify.factory

[filter:authorizer]
paste.filter_factory = nova.api.ec2:Authorizer.factory

[filter:validator]
paste.filter_factory = nova.api.ec2:Validator.factory

[app:ec2executor]
paste.app_factory = nova.api.ec2:Executor.factory

##############
# Openstack #
##############

[composite:osapi_compute]
use = call:nova.api.openstack.urlmap:urlmap_factory
/: oscomputeversions
/v1.1: openstack_compute_api_v2
/v2: openstack_compute_api_v2

[composite:osapi_volume]
use = call:nova.api.openstack.urlmap:urlmap_factory
/: osvolumeversions
/v1: openstack_volume_api_v1

[composite:openstack_compute_api_v2]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap noauth ratelimit osapi_compute_app_v2
deprecated = faultwrap auth ratelimit osapi_compute_app_v2
```

```
keystone = faultwrap authtoken keystonecontext ratelimit
    osapi_compute_app_v2

[composite:openstack_volume_api_v1]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap noauth ratelimit osapi_volume_app_v1
deprecated = faultwrap auth ratelimit osapi_volume_app_v1
keystone = faultwrap authtoken keystonecontext ratelimit
    osapi_volume_app_v1

[filter:faultwrap]
paste.filter_factory = nova.api.openstack:FaultWrapper.factory

[filter:auth]
paste.filter_factory = nova.api.openstack.auth:AuthMiddleware.factory

[filter:noauth]
paste.filter_factory = nova.api.openstack.auth:NoAuthMiddleware.
    factory

[filter:ratelimit]
paste.filter_factory = nova.api.openstack.compute.limits:
    RateLimitingMiddleware.factory

[app:osapi_compute_app_v2]
paste.app_factory = nova.api.openstack.compute:APIRouter.factory

[pipeline:oscomputeversions]
pipeline = faultwrap oscomputeversionapp

[app:osapi_volume_app_v1]
paste.app_factory = nova.api.openstack.volume:APIRouter.factory

[app:oscomputeversionapp]
paste.app_factory = nova.api.openstack.compute.versions:Versions.
    factory

[pipeline:osvolumeversions]
pipeline = faultwrap osvolumeversionapp

[app:osvolumeversionapp]
paste.app_factory = nova.api.openstack.volume.versions:Versions.
    factory

##########
# Shared #
##########

[filter:keystonecontext]
```

```
paste.filter_factory = nova.api.auth:NovaKeystoneContext.factory

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = keystone.openstack.b1-systems.de
service_port = 5000
auth_host = keystone.openstack.b1-systems.de
auth_port = 35357
auth_protocol = http
auth_uri = http://keystone.openstack.b1-systems.de:5000/
admin_tenant_name = b1systems
admin_user = admin
admin_password = testing
```

/etc/nova/nova.conf

```
[DEFAULT]
verbose=true
debug=true
api_paste_config=/etc/nova/api-paste.ini
osapi_compute_extension=nova.api.openstack.compute.contrib.
    standard_extensions
sql_connection=mysql://nova:testing@database.openstack.b1-systems.de/
    nova
rabbit_host=rabbitmq.openstack.b1-systems.de
rabbit_password=testing
image_service=nova.image.glance.GlanceImageService
glance_api_servers=glance-api.openstack.b1-systems.de:9292
auth_strategy=keystone
instances_path=/var/lib/nova/instances
connection_type=libvirt
libvirt_type=xen
instance_name_template=instance-%08x
```

Horizon

/usr/share/openstack_dashboard/local/local_settings.py

```
import os

from django.utils.translation import ugettext_lazy as _

DEBUG = True
TEMPLATE_DEBUG = DEBUG
PROD = False
```

```
USE_SSL = False

# Note: You should change this value
SECRET_KEY = 'elj1IWiLoWHgcyYxFVLj7cM5rGOOxWl0'

# Specify a regular expression to validate user passwords.
# HORIZON_CONFIG = {
#     "password_validator": {
#         "regex": '.*',
#         "help_text": _("Your password does not meet the
    requirements.")
#     }
# }

LOCAL_PATH = os.path.dirname(os.path.abspath(__file__))

# We recommend you use memcached for development; otherwise after
    every reload
# of the django development server, you will have to login again. To
    use
# memcached set CACHE_BACKED to something like 'memcached
    ://127.0.0.1:11211/'
CACHE_BACKEND = 'memcached://127.0.0.1:11211/'

# Send email to the console by default
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
# Or send them to /dev/null
#EMAIL_BACKEND = 'django.core.mail.backends.dummy.EmailBackend'

# Configure these for your outgoing email host
# EMAIL_HOST = 'smtp.my-company.com'
# EMAIL_PORT = 25
# EMAIL_HOST_USER = 'djangomail'
# EMAIL_HOST_PASSWORD = 'top-secret!'

# For multiple regions uncomment this configuration, and add (
    endpoint, title).
# AVAILABLE_REGIONS = [
#     ('http://cluster1.example.com:5000/v2.0', 'cluster1'),
#     ('http://cluster2.example.com:5000/v2.0', 'cluster2'),
# ]

OPENSTACK_HOST = "keystone.openstack.b1-systems.de"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"

# The OPENSTACK_KEYSTONE_BACKEND settings can be used to identify the
# capabilities of the auth backend for Keystone.
```

```
# If Keystone has been configured to use LDAP as the auth backend
    then set
# can_edit_user to False and name to 'ldap'.
#
# TODO(tres): Remove these once Keystone has an API to identify auth
    backend.
OPENSTACK_KEYSTONE_BACKEND = {
    'name': 'native',
    'can_edit_user': True
}

# The number of Swift containers and objects to display on a single
    page before
# providing a paging element (a "more" link) to paginate results.
API_RESULT_LIMIT = 1000

# If you have external monitoring links, eg:
# EXTERNAL_MONITORING = [
#     ['Nagios','http://foo.com'],
#     ['Ganglia','http://bar.com'],
# ]

LOGGING = {
        'version': 1,
        # When set to True this will disable all logging except
        # for loggers specified in this configuration dictionary.
            Note that
        # if nothing is specified here and disable_existing_loggers
            is True,
        # django.db.backends will still log unless it is disabled
            explicitly.
        'disable_existing_loggers': False,
        'handlers': {
            'null': {
                'level': 'DEBUG',
                'class': 'django.utils.log.NullHandler',
                },
            'console': {
                # Set the level to "DEBUG" for verbose output logging
                    .
                'level': 'INFO',
                'class': 'logging.StreamHandler',
                },
            },
        'loggers': {
            # Logging from django.db.backends is VERY verbose, send
                to null
            # by default.
            'django.db.backends': {
```

```
                'handlers': ['null'],
                'propagate': False,
                },
            'horizon': {
                'handlers': ['console'],
                'propagate': False,
            },
            'novaclient': {
                'handlers': ['console'],
                'propagate': False,
            },
            'keystoneclient': {
                'handlers': ['console'],
                'propagate': False,
            },
            'nose.plugins.manager': {
                'handlers': ['console'],
                'propagate': False,
            }
        }
}
```