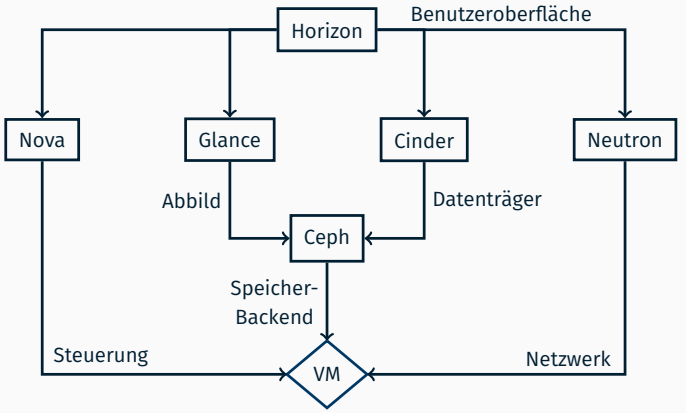
Cloud Computing
Software-Infrastruktur

Software-Infrastruktur am Beispiel von OpenStack



Software-Infrastruktur am Beispiel von OpenStack

- **Nova**: Verwaltung virtueller Maschinen
  - Compute: Steuerung von VMs (QEMU/Xen/...) auf Rechnern
  - Scheduler: Verteilung auf verfügbare Hardware
- **Glance**: Bereitstellung von Abbildern
  - Registry: Metadaten für Images
  - API unterstützt verschiedene Speichersysteme
- **Cinder**: Bereitstellung von Volumes
  - Volume-Service: Lokale Datenhaltung
  - Scheduler: Verteilung der Daten(-transfers) auf Rechner
- **Ceph**: Verteiltes Speicher-Backend für Glance und Cinder
- **Neutron**: Netzwerkmanagement und virtuelle Router
  - Server: Steuerung und Zustandsverwaltung
  - Agents: Helfer für DHCP, Open vSwitch, Metadaten
- **Horizon** (Dashboard): Weboberfläche für Anwender
- **REST-Schnittstelle**: API-Dienst je Komponente
  →Kommandozeilentools / SDK
- **RabbitMQ**: Interne Kommunikation der Dienste über Nachrichtenbus

Erstellen des Abbilds für OpenStack

Genereller Ablauf

- Ziel: Verlagerung der Übungsaufgabe in eine virtuelle Maschine
- Abbild innerhalb von OpenStack erzeugen
  - Starten einer Instanz des Linux-Live-System Grml (`http://grml.org`)
  - Neues Volume anlegen und einhängen
  - Betriebssysteminstallation
  - Anpassen der Konfiguration; Installieren zusätzlicher Softwarepakete
  - Umwandeln in Image
- Abbild starten
  - Öffentlichen SSH-Schlüssel für passwortlose Authentifizierung hinterlegen
  - Instanz mit eigenem Image starten
  - Öffentliche IP konfigurieren
  - Übungsaufgabe in der Cloud laufen lassen
- Speicherarten
  - Volume: Veränderbar, Verwendung nur in einer Instanz
  - Image (Abbild): Nicht veränderbar, Basis für viele Instanzen

Zugriff auf OpenStack

- Web-Frontend
  - Dashboard: `https://i4cloud1.cs.fau.de`
  - Zugangsdaten: siehe E-Mail mit Zugangsdaten
- Kommandozeile
  - OpenStack-Client-Programm: `openstack`
  - **Vor Verwendung**: `openrc`-Datei sourcen (siehe unten)
- Alle Kommandozeilenbefehle benötigen vorherige Authentifizierung
  1) Download der RC-Datei (`<user>-openrc.sh`) über Dashboard:
     → „Projekt“→ „API Access“→ „Download OpenStack RC File“
  2) RC-Datei einlesen und ausführen (sourcen)

```
$ source /path/to/<user>-openrc.sh
```
CIP

Grml-Instanz starten

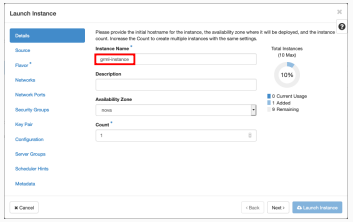
- Name für Instanz festlegen
- Instanztyp `i4.grml`

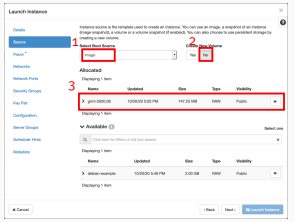
  → Kein Swap/Ephemeral-Volume
- Booten vom bereitgestellten Grml-Image (GRML-2020.06-amd64)

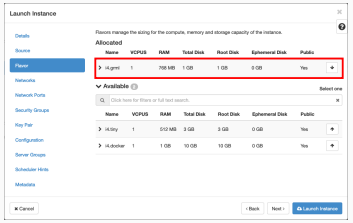
  → Kein zusätzliches Volume erzeugen
- Zugriff auf internes Netzwerk
- Weboberfläche: siehe nächste Folie
- Kommandozeile:

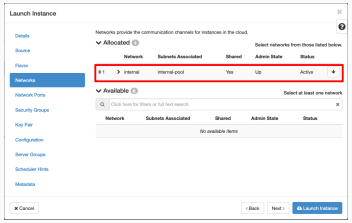
```
$ openstack image list    # --> grml id
$ openstack network list  # --> internal net id
$ openstack server create --flavor i4.grml \
    --image <grml id> \
    --nic net-id=<internal net id> \
    grml-instance
```
CIP

Grml-Instanz starten

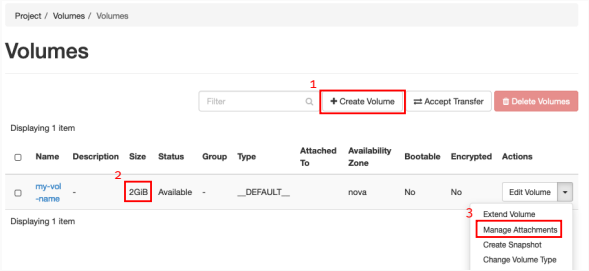








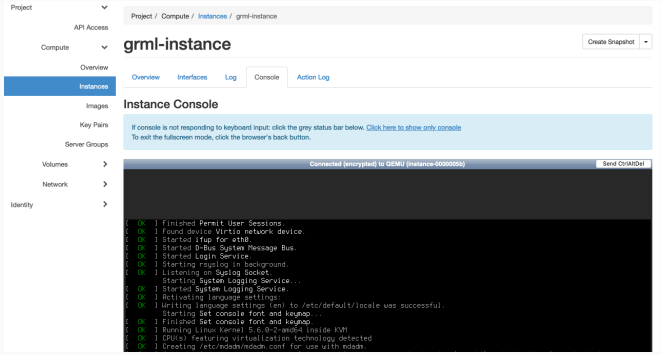
Volume erzeugen/einhängen



- (1) Leeres Volume anlegen, benötigt Name und Größe (2 GB)
- (2) Volumegröße kontrollieren
- (3) Volume der laufenden Instanz zuweisen
- Kommandozeile (Volume-Größe: 2 GB):

```
$ openstack volume create --size 2 my-vol-name  # --> vol ID
$ openstack server add volume grml-instance <vol id>
```
CIP

Erstellung des VM-Abbilds



- Konsole der laufenden Instanz im Dashboard öffnen
- Einrichtung des Betriebssystems und Installation der Java-Laufzeitumgebung im weiteren Verlauf der Übung

Erstellen einer Partition `parted(8)`, `mkfs(8)`

- Um als Basis für eine virtuelle Maschine zu dienen, muss das Abbild eine bootbare Partition mit Dateisystem beinhalten
- Mit `parted` lässt sich eine Partitionstabelle erstellen, was eine der Voraussetzungen ist, um das Abbild später booten zu können:

```
> parted /dev/vdb -s 'mktable msdos' 'mkpart primary 1MiB -1s' print
```
GRML

- Das Kommando `mkfs` (**m**ake **f**ile**s**ystem) erzeugt Dateisysteme, der Parameter `-t` spezifiziert dabei den Dateisystemtyp
- Erstellen eines `ext4`-Dateisystems mit der Bezeichnung „`VM-Abbild`" auf dem blockorientierten Gerät (block device) `/dev/vdb1`:

```
> mkfs -t ext4 -L "VM-Abbild" /dev/vdb1
```
GRML

Einhängen und Bootstrapping der Partition `mount(8)`, `debootstrap(8)`

Installation der User-Space-Komponenten des zukünftigen Gastbetriebssystems in das neu erzeugte, leere Dateisystem:

1. Einhängen des zuvor erstellten Dateisystems mit `mount`:

```
> mount /dev/vdb1 /mnt
```
GRML

Kontrolle:

```
> mount | grep vdb1
```
GRML

2. Erstellung der User-Space-Komponenten des Zielsystems mit `debootstrap`:

```
> debootstrap buster /mnt/ 'http://ftp.fau.de/debian'
```
GRML

Kontrolle:

```
> ls -alR /mnt | more
```
GRML

3. Setupskript mittels `wget` herunterladen und ausführbar machen:

```
> wget https://i4mw.cs.fau.de/openstack/post-debootstrap.sh -O /mnt/post-debootstrap.sh
> chmod +x /mnt/post-debootstrap.sh
```
GRML

Exkurs: Wechsel des Wurzelverzeichnisses — chroot(2)/(8)

- Jeder Linux-Prozess besitzt ein Wurzelverzeichnis (/)
  - Zugriff auf Daten außerhalb des Wurzelverzeichnisses ist **nicht** möglich
  - Kindprozesse erben das Wurzelverzeichnis ihres Elternprozesses (fork(2))
- Beispiel-Code jail.c:

```c
int main(int argc, char *argv[])
{
    /* Starte Kindprozess (/bin/bash) nach erfolgreichem
    Wechsel des Wurzelverzeichnisses */
    if (chroot("/mnt/") == 0) {
        execl("/bin/bash", NULL);
    }

    return 0;
}
```

- Die Datei /mnt/bin/bash des Live-Systems entspricht der Datei /bin/bash des Kindprozesses nach Aufruf von chroot(2)

Systemkonfiguration

- Weitergeben von /dev ins chroot (notwendig für die Installation von GRUB (Bootloader) im post-debootstrap.sh-Skript)

```
> mount -o bind /dev /mnt/dev
```
GRML

- Wechsel in das von debootstrap erstellte System mittels chroot(8)

```
> chroot /mnt /bin/bash
```
GRML

  ↪ **Hinweis:** Sämtliche **Änderungen** an dem von debootstrap erstellten System in der chroot-Umgebung sind **persistent**

- Aufruf des post-debootstrap.sh-Skriptes (siehe Aufgabenstellung) für grundlegende VM-Abbild-Konfiguration in der chroot-Umgebung und Setzen des Passworts für User cloud

```
# sh post-debootstrap.sh
Setting up /etc/apt/sources.list
(...)
Please set a password for user 'cloud'.
```
CHROOT

```
# passwd cloud
```
CHROOT

Software-Installation — apt-get(8)

- Ergänzen der Software des Grundsystems mittels apt-get
- Aktualisieren der Paketquellen (update) und anschließendes Einspielen potentiell vorhandener Updates (upgrade)

```
# apt-get update
# apt-get upgrade
```
CHROOT

- Das Kommando apt-get install löst Abhängigkeiten auf und installiert die entsprechenden Pakete, apt-get clean löscht Caches

```
# apt-get install <paket1> <paket2> ... <paketn>
# apt-get clean
```
CHROOT

- Für die Übung sind noch folgende Pakete nötig oder nützlich:

```
openssh-server openjdk-11-jdk-headless screen vim-nox
```
CHROOT

Ausführen der Java-Anwendung

- Installation benötigter Bibliotheken

```
# mkdir -p /proj/lib
# wget https://i4mw.cs.fau.de/openstack/libs.tgz -O libs.tgz
# tar -xvf libs.tgz -C /proj/lib
# rm libs.tgz
```
CHROOT

- Automatisches Starten der Dienste
  - Beim Systemstart führt systemd(1) die Init-Skripte aus
  - Bereitgestelltes Startskript /etc/systemd/system/i4mw-service.service
    1. Wertet Konfigurationsdaten (user-data) aus; siehe Aufgabenstellung
    2. Lädt jar-Datei mit der Anwendung aus S3 herunter
    3. Startet die Anwendung mit den angegebenen Parametern
- Hilfestellung zum Debugging
  - Ausgabe im Log der VM-Instanz beachten (per Dashboard einsehbar)
  - Ausgabe ist innerhalb der VM-Instanz im Syslog verfügbar

```
$ sudo less /var/log/syslog
```
VM

  - Nur die Ausgaben des Dienstes i4mw-service anzeigen

```
$ sudo journalctl -u i4mw-service
```
VM

VM-Umgebung verlassen und Abbild erzeugen

- Shell beenden, um `chroot`-Umgebung zu verlassen

```
# exit
```
CHROOT

- Grml-Live-Umgebung herunterfahren

```
> shutdown now
```
GRML

  - Eingehängte Dateisysteme werden automatisch ausgehängt
  - Stellt sicher, dass alle Änderungen geschrieben wurden
- Volume aushängen
  - Per Dashboard: „Volumes“→ „Manage Attachments“→ „Detach Volume“
  - Per Kommandozeile:

```
$ openstack server remove volume grml-instance <vol-id>
```
CIP

- Abbild erzeugen
  - Per Dashboard: „Volumes“→ „Upload to Image“, Imagenamen eingeben, Disk format auf `raw` setzen
  - Per Kommandozeile:

```
$ openstack image create --disk-format raw --volume <volume_id> <image_name>
```
CIP

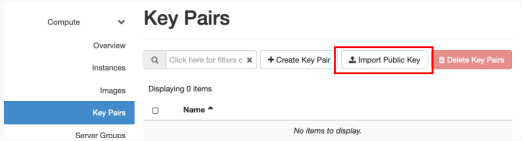
Betrieb der virtuellen Maschine

SSH-Schlüssel einrichten (einmalig)

- Privaten und öffentlichen Schlüssel mit `ssh-keygen` **auf einem CIP-Pool-Rechner** erzeugen

```
$ ssh-keygen -f ~/<gruppen_name> -N ""
Generating public/private rsa key pair.
Your identification has been saved in <gruppen_name>.
Your public key has been saved in <gruppen_name>.pub.
(...)
```
CIP

- Neu erstellten **öffentlichen Schlüssel** (`<gruppen_name>.pub`) hinzufügen unter „Compute“→ „Key Pairs“→ „Import Public Key“



- Kommandozeile:

```
$ openstack keypair create --public-key <gruppen_name>.pub <schluessel_name>
```
CIP

Eigenes Abbild als VM starten

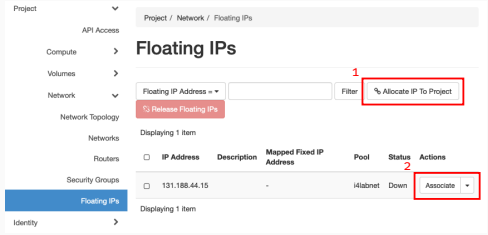
- Instanztyp `i4.tiny`
  → Erzeugt Swap-Disk und vergößert root-Partition
- Von eigenem Abbild starten
- SSH-Schlüssel unter „Key Pair“ auswählen

↪ Schlüssel wird beim Instanzstart nach `/home/cloud/.ssh/authorized_keys` kopiert

- Kommandozeile: (Schlüsselübergabe mittels Parameter `--key-name`)

```
$ openstack network list  # --> internal net id
$ openstack keypair list  # --> schluessel_name
$ openstack server create --flavor i4.tiny \
    --image <image name> \
    --nic net-id=<internal net id> \
    --key-name <schluessel_name> \
    my-vm-instance
```
CIP

Öffentliche IP zuweisen



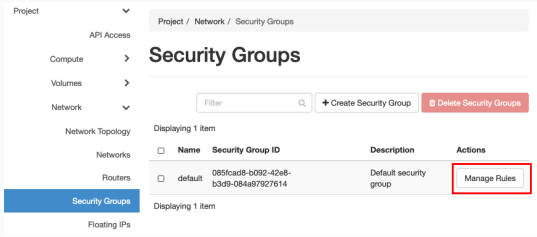
- (1) Öffentliche IP aus Pool allokieren, **nur einmalig nötig**
- (2) IP-Adresse an laufende Instanz zuweisen
- Kommandozeile:

```
$ openstack floating ip create i4labnet
$ openstack server add floating ip my-vm-instance <erhaltene IP>
```
CIP

- Abfrage innerhalb laufender VM per REST-API:

```
$ curl http://169.254.169.254/latest/meta-data/public-ipv4
```
VM

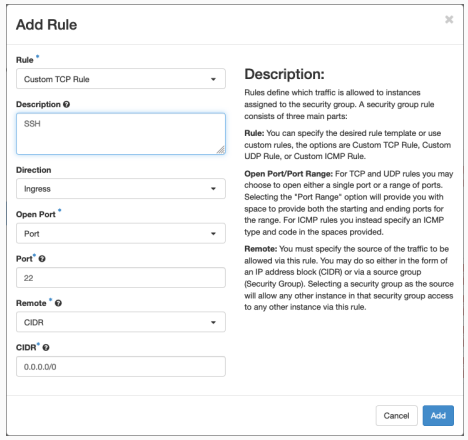
Zugriffsregeln für Netzwerkverbindungen



- TCP-Ports müssen für öffentlichen Zugriff freigegeben werden
- Kommandozeile, z. B. für TCP-Port 22 (SSH):

```
$ openstack security group rule create default \
    --ingress --src-ip 0.0.0.0/0 \
    --protocol tcp --dst-port 22
```
CIP

Firewall-Zugriffsregeln



Ingress = Eingehende Verbindungen, Egress = Ausgehende Verbindungen

Betrieb der virtuellen Maschine

- Passwortloser Zugriff mit SSH

```
$ ssh -i <gruppen_name> cloud@<instanz_ip>
```
CIP

→ Schlüssel aus der Rechnerübung, Instanz-IP aus vorheriger Zuweisung

- Wechsel von VM-Images erfordert evtl. Zurücksetzen von Host-Key

```
$ ssh-keygen -R <instanz_ip>    # Alten Host-Key entfernen
```
CIP

- Instanzen beenden: „Terminate“ auf der Weboberfläche, oder

```
$ openstack server list   # id heraussuchen
$ openstack server delete <instanz id>
```
CIP

- Alte Abbilder/Volumes löschen: Weboberfläche, oder

```
$ openstack volume delete <volume id>
$ openstack image delete <image id>
```
CIP

Nachträgliche Anpassungen am Abbild

- Neue GRML-Instanz starten und Volume einhängen (siehe Folie 5)
- Partition mit VM-Abbild mounten

GRML
```
> mount /dev/vdb1 /mnt
> mount -o bind /dev /mnt/dev
> chroot /mnt /bin/bash
```

CHROOT
```
# mount -t proc proc /proc
# mount -t sysfs sysfs /sys
# mount -t devpts devpts /dev/pts
```

- Volume anpassen
- GRML-Instanz ordentlich beenden

CHROOT
```
# exit
```

GRML
```
> shutdown now
```

- Volume aushängen und Abbild erneut hochladen (siehe Folie 15)

Private-Cloud-Umgebung des Lehrstuhls

- Modifikationen des VM-Abbilds über Grml-Instanz
  - Installation weiterer Softwarepakete
  - Anpassung der Startskripte
  - Systemkonfiguration
- Limitationen der Cloud-Umgebung des Lehrstuhls
  - Ressourcen der drei Node-Controller sind **beschränkt**
  - Beenden von nicht (mehr) benötigten Instanzen
  - Jederzeit auf faire Verwendung achten
- Infrastruktur
  - Bitte sendet bei Problemen oder Ungereimtheiten schnellstmöglichst eine E-Mail an `i4mw-owner@lists.cs.fau.de`

Anhang

Entwicklung eines VM-Abbilds

`dd(1), truncate(1)`

Hinweis: Im Folgenden gezeigte (Code-)Beispiele dienen als zusätzliche Information und sind für das Lösen der Übungsaufgabe nicht vonnöten.

- Gebräuchliche Abbild-Typen für virtuelle Maschinen (VM)
  - Kopie eines Datenträgers (z. B. ISO-Image einer CD oder DVD):

```
$ dd if=/dev/sdb of=./cd-image.iso
$ file -b ./cd-image.iso
ISO 9660 CD-ROM filesystem data (bootable)
```

  - Erzeugen einer leeren Abbild-Datei:

```
$ truncate -s 100M image.raw
$ ls -lh image.raw
-rw-r--r-- 1 thoenig users 100M  4. Nov 12:11 image.raw
$ du image.raw
0
$ file -b image.raw
data
```

- Alternativ ist es möglich, einen physischen Datenträger als Basis für eine virtuelle Maschine zu verwenden

Entwicklung eines VM-Abbilds — qemu(1)

- Die Erstellung und Aufbereitung des Abbilds der virtuellen Maschine benötigt erweiterte Privilegien (Root-Rechte)
- Die Aufbereitung des Abbilds geschieht daher isoliert in der Betriebsumgebung einer virtuellen Maschine („Live-System“)
  ↪ In der Übung: Linux-Live-System Grml (`http://grml.org`)
- Varianten, dieses Live-System zu verwenden
  - Mit Emulator `qemu`:

```
$ qemu -drive file=grml.iso,index=0,media=cdrom \
    -drive file=image.raw,index=1,media=disk
```

    [`root`-Dateisystem (Teil von `grml.iso`, Gerätepfad `/dev/sr0`) wird automatisch eingehängt, nicht jedoch das leere Abbild (`image.raw`, Gerätepfad `/dev/sda`)]
  - **In der Übung:** Instanz eines Grml-Abbilds direkt in der Cloud starten ↪ siehe vorangegangene Folien
- Eingerichtetes Abbild kann in Cloud hochgeladen werden

```
$ openstack image create --disk-format qcow2 \
    --file <image_file (e.g., image.raw)> <image_name>
```