

Inhaltsangabe

1. Allgemein	1
2. Erledigt, In progress, Probleme, Nodes.....	2
3. Projekthälfte Monitoring, Automatisierung, Selbstheilung.....	4

Ziel: Cloud Computing System mit integrierten Selbstheilungskonzepten










- OpenStack
- mögliche Fehlerfälle in einer Cloud Computing Umgebung analysieren
- Entwicklung eines Prozess, der Daten aus dem System sammelt, analysiert und im Fehlerfall reagiert

Produkt: Cloud Plattform (OpenStack), auf dem Nutzer Applikationen Hosten können (Bereitstellung einer Cloud-Lösung für Nutzer)

Empfehlung: kolla-ansible

- auf den 4 Servern soll OpenStack laufen
- Nutzer kann VMs starten
- Nutzer kann virtuelle Netzwerke nutzen
- Nutzer kann VMs von „außen“ erreichen (erstmal vom OpenStack Controller Node)

NOCH 5 WOCHEN

1. Gruppenbildung & Organisation	
2. Vortrag über ein Thema	
3. Gruppenorganisation & Zugriffsbereitstellung	
4. Planung	
5. Ergebnispräsentation	
6. Planung	
7. Meilensteinpräsentation	
8. Planung	
9. Ergebnispräsentation	
10. Planung	
11. Abschlusspräsentation	18.02.

Erledigt:

1. VPN-Konfigurationsdatei erhalten

- mit VPN-Client können wir jetzt mit dem Uni-Subnetz verbinden und auf die Server zugreifen

z.B. mit openvpn CLI:

```
'sudo openvpn -config /Pfad/zur/VPNKonfig.ovpn'
```

2. Unseren Gruppennamen festgelegt: OurSky

- Repository auf GitHub erstellt
- Rollen im Team:

Mitglieder und Rollen

- Jonathan (Scrum master & developer)
- Oliver (developer)
- Nadia (developer)
- Zead (Product owner & developer)

ssh key-Paar erstellt und an Alexander Public Key gesendet

OpenStack-Rollen: Admin & Nutzer

3. Vorbereitung Control Node
4. Vorbereitung Compute Node
5. Etablierung von Docker als Untersystem für OpenStack
6. Aktivierung & Konfiguration physischer Netzwerke
7. Bootstrapping
8. Prechecks
9. OpenStack Deployment



In progress:

Research – Wie geht es weiter?

Probleme:

Nodes:

auf wally135.cit.tu-berlin.de Controller Node (kolla-ansible)

auf wally139.cit.tu-berlin.de (OpenStack-ansible)

für Skript-Ausführung beiseite gelegt:

einzelne Skripte testen:

wally141.cit.tu-berlin.de Compute Node 1

ein Skript anwenden auf:

wally142.cit.tu-berlin.de Compute Node 2

In allen Nodes:

1. Interfaces in allen Nodes eingerichtet (/etc/netplan/01-netcfg.yaml)
2. Interfaces in /etc/network/interfaces konfiguriert
3. Hosts konfiguriert /etc/hosts → 10.0.42.135: controller, 10.0.42.141: compute1, 10.0.42.142: compute2
4. Kommunikation zwischen den Nodes verifiziert.
5. pip installiert
6. python-pip installiert
7. python-dev
8. libffi-dev
9. gcc
10. libssl-dev
11. chrony installiert und eingerichtet (NTP)
12. /etc/chrony.conf konfiguriert (compute Nodes sehen nur controller und controller erlaubt dies)

In Control Nodes:

1. Ansible installiert
2. kolla-ansible installiert
3. /etc/kolla erstellt
 - sudo chown \$USER:\$USER /etc/kolla
4. /usr/local/share/kolla-ansible/etc_examples/kolla/ in /etc/kolla kopiert
 - sf/usr/local/share/kolla-ansible/ansible/inventory/*
5. Im Haupt-Verzeichnis: multiple und all-in-node kopiert
 - /usr/local/share/kolla-ansible/ansible/inventory/*
6. /etc/ansible/ansible.cfg konfiguriert
7. Inventory: multiple konfiguriert
8. multiple nodes Erreichbarkeit verifiziert
9. Passwort in kolla-ansible erstellt: kolla-genpwd
10. docker installiert
 - Kolla globals.yml konfiguriert:
 - kolla_base_distro: „Ubuntu“
 - kolla_install_type: „binary“
 - networkt_interface: „eno1“
 - neutron_external_interface: „enp2s0“
 - kolla_internal_vip_address: „10.0.42.135“
11. bootstrap-servers in multiple ausgeführt
12. docker konfiguriert
13. verifiziert, ob Docker funktioniert
14. multiple konfiguration: alle Nodes local gesetzt

2. Projekthälfte Monitoring, Automatisierung, Selbstheilung

Weg 1:

Ihr bleibt bei der Cloud-Provider-Sicht und stattet eure OpenStack Cloud mit Monitoring aus. Das Monitoring soll sich dabei auf die wally-Server und die OpenStack Container richten. **Nicht** auf VMs oder Services die in der Cloud laufen. Ceilometer ist dabei also der Service, den man **nicht** nutzen sollte.

In den globals.yml gibt es diverse Parameter. Darunter z.B. central logging, kibana, prometheus. Setzt euch damit auseinander.

Weg 2:

Ihr wechselt die Sicht und seid nun der Admin und Nutzer der OpenStack Cloud. Ihr wollt dort euren Webservice "SockShop" in einem Kubernetes Cluster aufsetzen.

Dazu stelle ich euch Skripte bereit, mit denen ihr ein OpenStack-Netzwerk als Admin für alle Nutzer erstellt. Das lässt sich mit einem Nutzernetzwerk kombinieren. Den VMs können dann IPs aus der 10.0.42er Range zugewiesen werden, die dann von eurem lokalen Rechner aus erreichbar sind und selber Zugriff aufs Internet haben.

Ihr erzeugt einige Ubuntu-VMs, auf denen ein Kubernetes mit [kubespary](https://github.com/kubernetes-sigs/kubespary) (https://github.com/kubernetes-sigs/kubespary) installiert werden soll. In diesem Kubernetes wird dann der [SockShop](https://github.com/microservices-demo/microservices-demo) (https://github.com/microservices-demo/microservices-demo) Webservice ausgerollt. Dieser stellt Monitoring bereit (guckt euch dazu die Infos innerhalb der Repo an).

zu Weg 1)

- NICHT Ceilometer
- Panko - Event, Metadata Indexing Service
- Monasca - Monitoring

zu Weg 2)

- „SockShop“ in Kubernetes Cluster
- Admin erstellt für alle Nutzer ein OpenStack-Netzwerk
- Nutzernetzwerk
- VMs von unserem lokalen Rechner aus erreichbar
- Vms haben Zugriff aufs Internet

1. Ubuntu-VMs erzeugen

1. Allocate Floating IP to OpenStack
2. Create OpenStack Image
 1. Ubuntu in einer VM installieren
 2. Datei in raw konvertieren
 3. raw-Datei in OpenStack hochladen
 4. cloud-ubuntu Deployment

2. kubespary auf Ubuntu-VMs installieren

<https://medium.com/@iamalokpatra/deploy-a-kubernetes-cluster-using-kubespray-9b1287c740ab>

3. SockShop Webservice auf Kubernetes ausrollen -> stellt Monitoring bereit

<https://github.com/microservices-demo/microservices-demo/tree/master/deploy/kubernetes>