Kubernetes-Cluster mit Kubespray deployen

Strukturgeber

Tilman Beitter,
Christian Berendt

Frameworks zur Kubernetes-Bereitstellung und -Pflege gibt es mittlerweile viele. Kubespray basiert auf Ansible Playbooks und erschließt Anwendern den Nutzen von Kubernetes – unabhängig von der gewählten Cloud-Umgebung.

Kubernetes, kurz K8s, ist ein System zur automatischen Bereitstellung, Skalierung und Verwaltung containerbasierter Anwendungen. Ein manuelles Deployment von K8s auf einem Satz virtueller Systeme ist schnell durchgeführt. Eine einsatzbereite Umgebung steht mit relativ wenig Aufwand zur Verfügung. Doch denkt man meist im Vorfeld nicht darüber nach, wie man die Umgebung später pflegt, wie Scale-ups und Scale-downs zu handhaben sind und wie man ein Upgrade oder eine Konfigurationsänderung durchführt. Denn der Aufbau dient nur Testzwecken und wird später sicher nicht in Produktion gehen. Schließlich hat man eine historisch gewachsene Umgebung vor sich, die mit viel Liebe und Handarbeit zu pflegen und am Leben zu halten ist. Die Zeit, im Nachgang alles ordentlich aufzubauen, wird immer fehlen.

Dabei liegt es gerade bei einem System wie Kubernetes auf der Hand, ein Framework zu nutzen, um die Bereitstellung, Skalierung und Verwaltung zu automatisieren. Denn Kubernetes ist ja selbst ein System, das Automatisierungen auf Anwendungsebene bereitstellt. Warum soll man es manuell aufbauen und Fehler begehen, die man doch eigentlich durch die Einführung des Systems loswerden wollte.

Da Tools und Frameworks zum Deployment und zur Pflege von Kubernetes gerade wie Pilze aus dem Boden schießen, sollte man sich vor der Wahl einer Software etwas näher mit ihr und der Anzahl der an der Entwicklung beteiligten Entwickler und Firmen befassen. Gemäß der Devise „Do one thing and do it well“ sollte sich ein Tool für das Deployment und die Wartung von Kubernetes eben auch nur genau darum kümmern. Es sollte offen verfügbar sein, um bei nicht vorhersehbaren Aktionen wie dem Einstellen eines Projekts wenigstens noch Zugriff auf die Quellen zu bieten. Wünschenswert sind darüber hinaus die Möglichkeit einer flexiblen Integration sowie weitverbreitete und standardisierte Tools als Basis. Auch sollte es wenige Anforderungen an die zu nutzende Distribution oder die darunterliegende Infrastruktur haben.

Das Deployment-Framework Kubespray erfüllt diese Erwartungen: Die Zahl der mitwirkenden Entwickler ist seit 2015 auf über 400 angewachsen, der alleinige Fokus liegt auf Deployment und Pflege von Kubernetes. Die unter der Apache License 2.0 stehende Software nutzt Ansible und Terraform und lässt sich mit vielen unterschiedlichen Distributionen und Clouds nutzen.

Test-Set-up in der OpenStack-Cloud

Ziel ist ein Basisaufbau der Umgebung auf einer OpenStack-basierten Cloud. Die gewählte Distribution ist Ubuntu 18.04, doch auch CentOS, RHEL, openSUSE oder Debian sind möglich. Als Containerumgebung dient Docker, flannel stellt das Netzwerk für Kubernetes zur Verfügung. Für das Deplyoment der Ressourcen wie Instanzen, Netzwerke und Volumes setzt Kubespray unter anderem auf Terraform, ein Werkzeug zum Aufbau und zur Pflege virtualisierter Infrastrukturen. Konkrete Umsetzungen für OpenStack, AWS und Packet existieren bereits. Im Beispiel dieses Artikels kommt der Provider für OpenStack zum Einsatz, da die benötigte Infrastruktur auf einer OpenStack-basierten Cloud aufzubauen ist.

Der Aufbau startet zu Beginn mit einem Master und zwei Workern, später kommen weitere Worker hinzu (siehe Abbildung). Um administrative Aufgaben und Zugänge von der Workload (dem Zugriff der Anwender auf die von Kubernetes bereitgestellten Services) zu trennen, wird zusätzlich ein Bastion-Host aufgesetzt. Er erhält eine öffentliche IP-Adresse für den externen Zugriff via SSH und erreicht die Nodes über ein internes Netzwerk. Er dient als Jump-Host und stellt den SSH-Zugang und damit auch den Zugriff für Ansible auf die Kubernetes-Systeme zur Verfügung. Für den Aufbau der Evaluationsumgebung

X-TRACT

- Kubespray ist ein Framework zum Bereitstellen und Verwalten von Kubernetes-Clustern.
- Es ist Open Source und steht unter der Apache-2.0-Lizenz.
- Mit Kubespray lassen sich Cluster skalieren, das heißt bei Bedarf verkleinern oder vergrößern

kann man darauf verzichten, den Nodes öffentliche IP-Adressen zu geben. Der Zugriff auf die dort bereitgestellten Services erfolgt durch einen Tunnel über den Bastion-Host.

Um Kubespray nutzen zu können, sind einige Grundvoraussetzungen zu erfüllen. Auf dem Administrationssystem, das eine dedizierte Instanz oder aber die eigene Workstation sein kann, müssen Terraform und Ansible installiert sein. Terraform bietet Downloads für viele Systeme an; für die Installation von Ansible lassen sich Pakete aus dem distributionseigenen Repository verwenden. Neben den Vorbereitungen auf dem Administrationssystem gibt es ein paar Punkte in der Cloud (hier OpenStack) zu beachten: Ein passendes Image (in unserem Fall Ubuntu 18.04) muss vorliegen, ein Floating IP Address Pool existieren und das Nutzen von Security Groups muss möglich sein.

Ordnerbezogene Installation und Konfiguration

Von einem dedizierten Administrationssystem im folgenden Set-up ausgehend, gestaltet sich die Installation von Kubespray einfach. Lediglich das offizielle Repository von GitHub klont man:

```
git clone https://github.com/kubernetes-sigs/ ↵
                                    kubespray.git
```

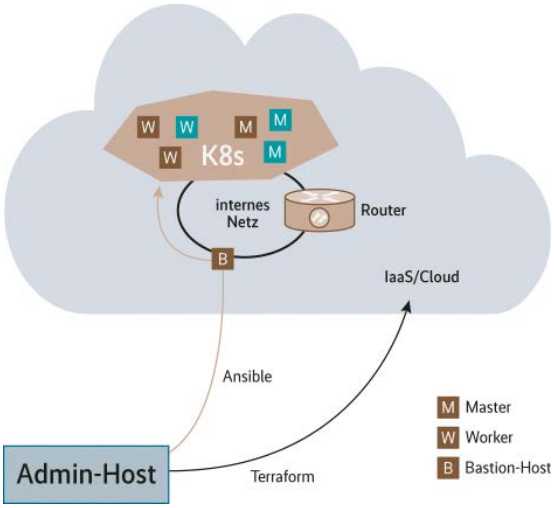
Hinweis: Bei den Aufrufen von Terraform und Ansible ist es notwendig, sich jeweils im richtigen Ordner zu befinden. Daher ist zu beachten, dass man alle terraform-Kommandos aus dem Inventory-Ordner (hier ./kubespray/inventory/ix) heraus startet und sich für den Start von ansible-playbook stets im Root-Verzeichnis des Kubespray-Repository (hier ./kubespray) befinden muss. Hierzu sind auch alle Pfadangaben relativ.

Der für die aufzubauende Umgebung im Repository-Klon angelegte Inventory-Ordner heißt ix, was zugleich der Name für den Cluster ist. In diesem Ordner lassen sich später alle für diese Umgebung erforderlichen Informationen wiederfinden. Kubespray stellt einen Sample-Ordner bereit, der als Vorlage dienen kann. Danach sind nur noch zwei Symlinks notwendig, um das OpenStack-Plug-in für Terraform sowie das Inventory-Plug-in für Ansible verfügbar zu machen:

```
cp -LRp contrib/terraform/openstack/↵
                sample-inventory inventory/ix
ln -s ../../contrib/terraform/openstack/ ↵
                            hosts inventory/ix
ln -s ../../ contrib inventory/ix
```

Der nächste Schritt zeigt das Erstellen eines Schlüsselpaars für SSH, mit dem später der Zugriff auf die bereitgestellten Systeme möglich ist:

```
ssh-keygen -t rsa -b 2048 -N "" ↵
 -C "" -f inventory/ix/id_rsa_ix
```



Die Testumgebung basiert auf einer OpenStack-Cloud. Ein Bastion-Host dient als Jump-Host, der den SSH-Zugang auf die Kubernetes-Systeme ermöglicht.

Damit Terraform weiß, welche Cloud es zum Bereitstellen der Ressourcen verwenden soll, ist die zu nutzende OpenStack-Umgebung zu konfigurieren. Terraform liest dafür die Datei clouds.yaml aus. Damit sich später sensitive Daten verschlüsselt speichern lassen, legt man sie in einer separaten secure.yaml-Datei ab. Die Verschlüsselung der secure.yaml soll nicht Gegenstand dieses Beitrags sein. Bitte beachten: Die Dateiendung .yaml ist relevant, die Kurzform .yml ignoriert Terraform (Listing 1; alle Listings zum Download unter ix.de/zk9k).

Genügend Ressourcen für Master und Worker Nodes

Dank der eben erstellten Konfiguration weiß Terraform nun, auf welcher Cloud der Cluster aufzubauen ist. Als Nächstes gilt es festzulegen, was Terraform dort erstellen soll. Zu diesem Zweck lässt sich die Datei cluster.tfvars nutzen, sie enthält alle relevanten Parameter für die definierten Ressourcen. Sie ist der Übersichtlichkeit halber in mehrere Blöcke unterteilt (Listing 2).

Im ersten Block konfiguriert man generelle Informationen über die Umgebung, etwa den Namen des Clusters oder das zu verwendende Image. Im zweiten und dritten Block stellt man ein, welche Anzahl an Systemen aufzubauen sowie in welcher Ausprägung (mit oder ohne

Listing 1: Cloud festlegen (inventory/ix/clouds.yaml)

```
clouds:
  mycloud:
    profile: betacloud
    auth:
      username: [...]
      password: mypass [...]
      user_domain_name: mydomain
      project_name: myprocject
      project_domain_name: mydomain
```

Listing 2: Variablen konfigurieren (inventory/ix/cluster.tfvars)

```
cluster_name = "ix"
az_list = ["south-1"]
az_list_node = ["south-1"]
public_key_path = "id_rsa_ix.pub"
image = "Ubuntu 18.04"
ssh_user = "ubuntu"

number_of_etcd = 0
number_of_k8s_masters = 0
number_of_k8s_masters_no_etcd = 0
number_of_k8s_masters_no_floating_ip = 1
number_of_k8s_masters_no_floating_ip_no_etcd =
0

flavor_k8s_master = "bd888..."
number_of_k8s_nodes = 0
number_of_k8s_nodes_no_floating_ip = 2
flavor_k8s_node = "bd888..."

network_name = "ix"
external_net = "0647c0..."
subnet_cidr = "192.168.0.0/24"
floatingip_pool = "public"

number_of_bastions = 1
flavor_bastion = "bd888..."
bastion_allowed_remote_ips = ["0.0.0.0/0"].
```

Listing 3: Cloud-Zuweisung und Ausgabe der Ressourcen

```
$ export OS_CLOUD=mycloud
$ terraform apply -var-filecluster.tfvars contrib/terraform/openstack

Apply complete! Resources: 20 added, 0 changed, 0 destroyed.
Outputs:
bastion_fips = [
    185.136.140.40
]
floating_network_id = 0647c0a0-862c-4c7e-9433-4558fcc5573b
k8s_master_fips = []
k8s_node_fips = []
private_subnet_id = a1670441-1312-4e61-abae-c7cf3d5a4f3f
router_id = 5a76861f-db7d-45d9-a93c-59e6813effaf
```

Floating IP Addresses) und mit welchen Ressourcen (Flavor für vCPUs, RAM und Festplatte) sie auszustatten sind. Für die Master und Worker Nodes sollte man mindestens 2 GByte Memory und 2 vCPUs nutzen. Der vierte Block enthält die Netzwerkkonfiguration. Hier definiert man das zu erstellende Netzwerk und gibt an, welche bereits existierenden externen Netze und Floating IP Address Pools zu nutzen sind. Den zuvor angesprochenen Bastion-Host definiert man im nächsten Block. Das System dient nur als SSH-Jump-Host und lässt sich daher dementsprechend klein dimensionieren.

Der bereits skizzierte Aufbau (siehe Abbildung) beinhaltet einen Master und zwei Worker ohne öffentliche IP-Adressen. Dazu sind die Variablen `number_of_k8s_masters_no_floating_ip` auf `1` und `number_of_k8s_nodes_no_floating_ip` auf `2` zu setzen. Zu beachten ist, dass man die anderen Ausprägungen, etwa ein Master mit Floating IP Address (`number_of_k8s_ masters`), auch wirklich auf `0` setzt, da der Default hier nicht `0` ist und sonst unerwünschte Systeme entstünden.

Um bequem vom Admin-Host aus den Cluster bedienen zu können, kann Kubespray den Konsolenclient kubectl lokal konfigurieren. Kubespray legt dazu im Ordner inventory/ix/artifacts ein Helper-Skript kubectl.sh, eine zum Cluster passende Konfiguration admin.conf und das kubectl-Binary an. Ansible definiert diese Einstellungen sowie das gewünschte Network-Plug-in innerhalb der `group_vars` in der Datei inventory/ix/group_vars/k8s-cluster/k8s-cluster.yml:

```
kubeconfig_localhost: true
kubectl_localhost: true
kube_network_plugin: flannel
```

Damit sind alle notwendigen Konfigurationen erledigt und der Clusteraufbau kann beginnen.

Terraform unterstützt Deployment

Im ersten Schritt ist die erforderliche Infrastruktur in der gewählten Cloud aufzubauen. Damit Terraform die Konfiguration und die Plug-ins finden kann, arbeitet man im Ordner inventory/ix und führt zuerst einen Init durch. Terraform installiert die benötigten Plug-ins:

```
terraform init contrib/terraform/ ⤷
                                openstack
```

Im nächsten Schritt teilt man dem System mit, auf welcher Cloud die Umgebung zu erstellen ist. Dieser Name muss mit der zuvor in der Datei clouds.yaml definierten Cloud übereinstimmen. Anschließend erfolgt der Start von Terraform, das Erstellen der Ressourcen wie Netzwerke, Router und Instanzen und die Ausgabe einer kurzen Übersicht (Listing 3).

Alle Vorbereitungen sind getroffen und die Systeme erstellt. Kubespray ist nun in der Lage, den Cluster aufzubauen. Dazu startet man Ansible und gibt das Inventory, den SSH-Key und das Playbook cluster.yml an:

```
ansible-playbook --become -i inventory/ix/ ⤷
                                        hosts -e
ansible_ssh_private_key_file=inventory/ix/id_ ⤷
                rsa_ix" cluster.yml --flush-cache
```

Die Abarbeitung des Playbooks und aller Rollen wird nun einige Zeit in Anspruch nehmen.

Der Cluster steht – was nun? Um zu zeigen, dass Kubernetes zu diesem Zeitpunkt voll funktionsfähig ist, deployt man einen Service. Kubernetes bietet dazu ein kurzes „Hello World"-Kubernetes-Set-up an. Da man eingangs dafür gesorgt hatte, kubectl und eine passende Konfiguration auf den Admin-Host zu kopieren, kann man den Cluster von diesem System aus bedienen. Um die Systeme hinter dem Bastion-Host erreichen zu können, ist im Set-up (ohne Public IP Address an den K8s-Systemen) das interne Netz verfügbar zu machen. Dazu nutzt man nach einem Wechsel ins Verzeichnis ~/kubespray das Tool sshuttle und installiert zuvor Python auf dem Bastion-Host:

```
ssh -i inventory/ix/id_rsa_ix ⤷
                ubuntu@BASTION_ADDRESS 'sudo ⤷
                      apt-get install python'
sshuttle -D -e 'ssh -i ⤷
                    inventory/ix/id_rsa_ix' ⤷
       -r ubuntu@BASTION_ADDRESS 192.168.0.0/24
```

Durch den Aufruf von get nodes erhält man einen Überblick über den Clusterzustand (Listing 4). Die Ausgabe zeigt, dass der Cluster wie gewünscht aufgebaut wurde und funktioniert. Es erfolgt das Erstellen des Testservice. Dazu erzeugt man ein Deployment mit zwei Instanzen:

```
$ ./kubectl.sh run hello-world --replicas=2 \
  --labels="run=load-balancer-example" \
  --image=gcr.io/google-samples/node- ⤷
                                  hello:1.0 \
  --port=8080
deployment.apps/hello-world created
```

Als Nächstes fertigt man einen Service an, der das Deployment auf einem lokalen Port zur Verfügung stellt. Für einen Test ist die Nutzung von NodePorts ausreichend:

```
$ ./kubectl.sh expose deployment hello-world ⤷
       --type=NodePort --name= example-service
service/example-service exposed
```

Ob die Pods korrekt starten, prüft man mit get pods (Listing 5).

Für den Test gilt es noch herauszufinden, welche Portzuweisung der Service example-service enthält. Das Kommando describe hilft (Listing 6). Es zeigt sich, dass der Service unter dem bei NodePort aufgeführten Port 32732/tcp zur Verfügung gestellt wurde. Der Dienst lässt sich mit einem curl-Aufruf testen:

```
$ curl http://MASTER_ADDRESS:32732
Hello Kubernetes!
```

Somit ist das Deployment des Kubernetes-Clusters abgeschlossen.

Auch wenn jegliche Konfiguration über Kubespray erfolgen sollte, ist es hin und wieder notwendig, sich selbst

Listing 4: Clusterzustand ermitteln

```
$ cd inventory/ix/artifacts
$ ./kubectl.sh get nodes
NAME                STATUS     ROLES      AGE       VERSION
ix-k8s-master-nf-1  Ready      master     27m       v1.13.3
ix-k8s-node-nf-1    Ready      node       26m       v1.13.3
ix-k8s-node-nf-2    Ready      node       26m       v1.13.3
```

Listing 5: Überprüfen der gestarteten Pods

```
$ ./kubectl.sh get pods
NAME                           READY     STATUS     RESTARTS  AGE
hello-world-696b6b59bd-7d8md   1/1       Running    0         3m24s
hello-world-696b6b59bd-bz64c   1/1       Running    0         3m24s
```

Listing 6: Mit describe den Port ermitteln

```
$ ./kubectl.sh describe services example-service
Name:                     example-service
Namespace:                default
Labels:                   run=load-balancer-example
Annotations:              <none>
Selector:                 run=load-balancer-example
Type:                     NodePort
IP:                       10.233.1.98
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32732/TCP
Endpoints:                <none>
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Listing 7: Playbook aufrufen und Clusterstatus prüfen

```
$ ansible-playbook --become become-user=root\
    --inventory=inventory/ix/hosts \
    -e "ansible_ssh_private_key_file=inventory/ix/id_rsa_ix" \
    scale.yml

$ cd inventory/ix/artifacts
$ ./kubectl.sh get nodes
NAME                STATUS    ROLES     AGE    VERSION
ix-k8s-master-nf-1  Ready     master    112m   v1.13.3
ix-k8s-node-nf-1    Ready     node      111m   v1.13.3
ix-k8s-node-nf-2    Ready     node      111m   v1.13.3
ix-k8s-node-nf-3    Ready     node      2m3s   v1.13.3
ix-k8s-node-nf-4    Ready     node      2m4s   v1.13.3
```

Listing 8: Einen Worker entfernen

```
$ ansible-playbook --become -i inventory/ix/hosts \
    -e ansible_ssh_private_key_file=inventory/ix/id_rsa_ix \
    -e node=ix-k8s-node-nf-4 remove-node.yml

$ cd inventory/ix/artifacts
$ ./kubectl.sh get nodes
NAME                STATUS   ROLES    AGE    VERSION
ix-k8s-master-nf-1  Ready    master   120m   v1.13.3
ix-k8s-node-nf-1    Ready    node     119m   v1.13.3
ix-k8s-node-nf-2    Ready    node     119m   v1.13.3
ix-k8s-node-nf-3    Ready    node     10m    v1.13.3
```

auf den Systemen einzuloggen, sei es, um Probleme zu debuggen oder rein aus Interesse. Man kann per SSH über den Bastion-Host springen oder, und das ist der Anspruch, von der Automatisierung durch Kubespray profitieren. Die beim Deployment mit Terraform erstellte Konfiguration für SSH ermöglicht den direkten Zugriff über den Bastion-Host auf die Systeme.

Notfall-Log-in via Bastion-Host

Um die Konfiguration nutzen zu können, kopiert man sie in den eigenen ~/.ssh-Ordner oder fügt den Inhalt in die bestehende Konfigurationsdatei ein. Da man sich in diesem Set-up auf einem dedizierten Admin-Host befindet, existiert keine über Jahre gepflegte SSH-Konfiguration: Die im Root-Verzeichnis des Repos generierte ssh-bastion.conf lässt sich direkt kopieren:

```
cp ssh-bastion.conf ~/.ssh/config
ssh -i inventory/ix/id_rsa_ix ↵
    ubuntu@192.168.0.11 ubuntu@ix-k8s-node-nf-1
```

Kubespray dient nicht nur dazu, die Umgebung initial aufzubauen, sondern kann auch Wartungsarbeiten am Cluster ausführen. Entspricht der initiale Aufbau zu einem Zeitpunkt nicht mehr den Anforderungen, weil der Service gut oder schlecht angenommen wurde, ist es notwendig, den Cluster zu vergrößern oder zu verkleinern.

Weitere Nodes lassen sich in wenigen Schritten hinzufügen. Zunächst teilt man Terraform die neue Anzahl an Systemen mit, indem man die inventory/ix/cluster.tfvars-Datei editiert:

```
# number_of_k8s_nodes_no_floating_ip = 2
number_of_k8s_nodes_no_floating_ip = 4
```

Terraform merkt sich beim Aufbau die Ressourcen und kann bei späteren Ausführungen die notwendigen Änderungen erkennen und durchführen:

```
terraform apply -var-file=cluster.tfvars ↵
                    contrib/terraform/openstack
```

Nachdem Terraform die neuen Systeme erstellt hat, ruft man das Playbook scale.yml auf und überprüft anschließend mit `kubectl get nodes` den Status des Clusters (Listing 7).

Mit Kubespray ist es kein Problem, den Cluster um Worker Nodes zu verkleinern – der Scale-down von Master oder etcd Nodes ist jedoch (noch) nicht möglich. Der Prozess läuft umgekehrt zum Scale-up. Zuerst entfernt man den Worker aus Kubernetes, danach lässt man Terraform die Ressourcen abbauen. Da Terraform die Systeme von 1 aufsteigend durchnummeriert, ist es wichtig, immer den oder die letzten Worker zu entfernen. Im vorliegenden Beispiel eliminiert man einen Worker (Listing 8).

Nachdem Kubespray das System aus dem Cluster beseitigt hat, editiert man die Datei inventory/ix/cluster.tfvars:

```
# number_of_k8s_nodes_no_floating_ip = 4
number_of_k8s_nodes_no_floating_ip = 3
```

Anschließend ruft man Terraform auf, um die Instanz auch von der Cloud zu nehmen:

```
$ terraform apply -var-file=cluster.tfvars
contrib/terraform/openstack
```

Neben der Skalierung der Umgebung ist mit dem Playbook upgrade.yml auch das Durchführen eines Upgrades der ganzen Umgebung oder einzelner Bestandteile möglich. Auch lässt sich Ansible erheblich beschleunigen, wenn man die Python Library Mitogen über das Playbook mitogen.yml aktiviert.

Fazit

Kubespray folgt prinzipiell der Unix-Philosophie „One tool, one job“ und kümmert sich nur um die Bereitstellung und Pflege von Kubernetes. Das Bereitstellen der notwendigen Ressourcen auf der Infrastruktur, das Monitoring und die Visualisierung übernehmen andere Werkzeuge wie Terraform. Das initiale Bereitstellen ist dadurch zeitlich aufwendiger, im Vergleich zu Lösungen wie OpenStack Magnum ist man aber deutlich breiter aufgestellt. Alle Bestandteile des Frameworks stehen zur Verfügung, was eine flexible Handhabung von Integration, Customizing und Bugfixing erlaubt.

Es soll aber nicht unerwähnt bleiben, dass es noch andere Deployment-Frameworks für Kubernetes gibt. Dazu zählt kops, das ebenfalls den Kubernetes-Einsatz auf unterschiedlichen Cloud-Lösungen ermöglicht und Thema eines *iX*-Artikels sein wird. (avr@ix.de)

Quellen

[1] Listings zum Aufbau einer auf OpenStack-basierten Umgebung: ix.de/zk9k

Tilman Beitter

ist bei B1 Systems Linux-Consultant und Trainer. Er kümmert sich um Support, Operations und Cloud-Computing, speziell OpenStack.

Christian Berendt

arbeitet als Solution Architect für die B1 Systems GmbH und betreut derzeit den Bereich „Cloud Computing“, beschäftigt sich aber auch intensiv mit Systemmanagement.