Kubernetes the not so hard way with Ansible (at Scaleway) - Part 1 - The basics

Ansible introduction and setup hosts at Scaleway

December 4, 2016

I'll create a series of posts about running Kubernetes (k8s for short) at Scaleway managed by Ansible. But you should be able to use the playbooks with minor modifications for other hoster. I'll only test this with Ubuntu 16.04 LTS but with minimal modifications it should work with all systemd based Linux operating systems. I used Kelsey Hightower's wonderful guide Kubernetes the hard way as starting point. My goal is to install a Kubernetes cluster with Ansible which could be used in production and is maintainable. It's not H/A at the moment because the Kubernetes components currently communicate only with one API server. So ATM there are still a few TODO's besides make API server H/A e.g. upgrading parts of the cluster will need some changes in the playbooks/roles later I guess. But my first goal is to get the cluster up and running.

If you need something fast maybe Minikube or kubeadm is more suited for you ;-)

Why Scaleway

As already mentioned this tutorial and the playbooks should work with other hoster like DigitalOcean too (maybe with some minor modifictions as long as you use Ubuntu 16.04 LTS) but I'll focus on Scaleway because I use it for quite a time now and I like it for some good reasons:

- Good pricing (starting from small VPS at €2.99/month with 50 GB SSD)
- Add more HDD SSD space for €1/month for 50 GB
- Locations in Paris and Amsterdam with good peering
- Unmetered transfer with bandwith starting at 200 MBit/s (depends on host type)
- API to manage resources and CLI tool
- Very stable (never had an issue during the last 12 month)
- Snapshots, volumes and images
- Docker kernel support (using a very recent kernel which is cool if you use the latest Docker releases)
- Hardware instances using ARM or x86_64 processor
- and some other goodies

Disclaimer: I'm not working for Scaleway and don't get paid ;-) So all in all a very good choise for a person like me ;-) You won't get supa dupa performance at Scaleway because most of the cheap instances run on Intel Atom CPUs but it's more than enough power for me… They also have bigger instances of course so don't worry (if they don't just be temporary out of stock…) But forget about the security groups. They're basically useless since you can't define a default rule (which should always be `deny` for basically every port - excluding SSH at the beginning which we need for Ansible). If you have to block every port that shouldn't be accessable from the internet you'll get crazy and that's not how security works. If you've a jumphost you can remove the public IPs from the other instances if they don't offer services to the outside like databases which saves you money too. I'll use ufw (which basically just manages Linux netfilter) to allow or deny access (as you'll see later).

To run the services on the same network I'll use PeerVPN. Kelsey Hightower uses Google Cloud or AWS which supports cool networking options but we don't have this features at Scaleway. So PeerVPN will help us to compensate this a little bit as we can create a network at layer 2 with communication encrypted and it's easy to install. But more about that in a later blog post. Another option would be WireGuard but it's not yet in the official Linux kernel (may happen during 2018 - for futher information: https://github.com/hobby-kube/guide#wireguard-setup).

Start your engines

I assume that you've already an account at Scaleway or whatever hoster you use. At Scaleway first add your public SSH key to the credentials page. This key will be added to `/root/.ssh/authorized_keys` during launch of a host. Since Ansible is using SSH for mananging a instance we use this key for Ansible to login (at least during the first login). For other SSH key options see Scaleway documentation. As a side note: The root password is stored in a file `/root/.pw` on every host. You shouldn't leave it there ;-) But my harden-linux role will take care of it.

If you want to do something real with your Kubernetes cluster you'll need at least 5 instances. Three for the Kubernetes controller nodes and for etcd (three nodes for high availbility) and two nodes for the worker (the nodes that will run the Docker container and do the actual work). So we'll launch 5 VC1S instances (2x64 bit core, 2 GB RAM, 50 GB SSD each) which should be suffcient for now. I try to keep costs low. So if you run production load you should distribute the services on more hosts and use bigger hosts for the worker (maybe something like C2L).

Scaleway provides a handy CLI tool which you can use for managing server. Since it's a tool written in Golang it's pretty easy to install. While writing this post the latest version was 1.14. So just download https://github.com/scaleway/scaleway-cli/releases/download/v1.14/scw-linux-amd64, copy the downloaded binary scw-linux-amd64 binary to e.g. /usr/local/bin/scw and make it executable (`chmod 0755 scw`). To use the CLI we first need to authenticate with `scw login` . Provide your login data and you're done.

Now we can create the server e.g.:

```
scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-controller1" --attach=true
Ubuntu_Xenial

scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-controller2" --attach=true
Ubuntu_Xenial

scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-controller3" --attach=true
Ubuntu_Xenial

scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-worker1" --attach=true Ubuntu_Xenial

scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-worker2" --attach=true Ubuntu_Xenial
```

`--region=ams1` will launch the server in Amsterdam data center (use `--region=par1` for Paris).

`--commercial-type=VC1S` specifies the host type we want to launch (actually the smallest VPS at Scaleway).

`--bootscript=38a358a0` will boot the instance with a Linux kernel that is needed for Docker (Kernel 4.14.12-rev1 in that case). The "mainline" bootscript now contains everything that ist needed for Docker. Before Kernel 4.14.11 you needed a special bootscript for Docker. To get the latest available "mainline" bootscript use `scw --region="ams1" images -f type=bootscript | grep x86_64_mainline` for Amsterdam region or use `... --region="par1" ...` for Paris region. Kernel >=4.14.12-rev1 also protects you from Meltdown (see https://twitter.com/scaleway/status/949441000941998080)

`--name=...` specifices a name for our instance for better identification.

The option `--attach=true` allows us to attach to the boot process (which in turn prints the public/private IPs and a few other information).

And finally we specify that we want launch Ubuntu 16.04 (`Ubuntu_Xenial`).

As a side note: We'll install the etcd cluster on the controller nodes (e.g. where the API server, K8s scheduler and K8s controller manager runs) to save costs. But it's recommended for production to install etcd on it's own hosts. So you may install three additional hosts:

```
scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-etcd1" --attach=true Ubuntu_Xenial

scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-etcd2" --attach=true Ubuntu_Xenial

scw --region="ams1" run --commercial-type=VC1S --bootscript=38a358a0 --name "k8s-etcd3" --attach=true Ubuntu_Xenial
```

Prepare Ansible

If you never heard of Ansible: Ansible is a powerful IT automation engine. Instead of managing and handling your instances or deployments by hand Ansible will do this for you. This is less error phrone and everything is repeatable. To do something like installing a package you create a Ansible task. This tasks are organized in playbooks. The playbooks can be modified via variables for hosts, host groups, and so on. A very useful feature of Ansible are roles. E.g. you want to install ten hosts with Apache webserver. In that case you just add a Apache role to that ten hosts and maybe modify some host group variables and roll out Apache webserver on all the hosts you specified. Very easy! For more information read Getting started with Ansible. But I'll add some comments in my blog posts what's going on in the roles/playbooks we use.

For beginners: Also have a look here: ANSIBLE BEST PRACTICES: THE ESSENTIALS

I was also thinking about using ImmutableServer and Immutable infrastructure but decided to go with Ansible for now. This concepts have some real advantages and we also using it in my company very successfully together with the Google Cloud. Using virtual machines like Docker container and throw them away at any time is quite cool :-) .

Setup Ansible

If you haven't already setup a Ansible directory which holds our hosts file, the roles and so on then do so now. The default directory for Ansible roles is `/etc/ansible/roles`. To add an additional roles directory adjust the Ansible configuration `/etc/ansible/ansible.cfg` and add your roles path to `roles_path` setting (separated by :). I'm a fan of having everything in one place so I put everything for Ansible in `/opt/ansible`. So the roles path for me is `roles_path /opt/ansible/roles:/etc/ansible/roles`. A tool for installing Ansible roles is `ansible-galaxy` which is included if you install Ansible. Also have a look at https://galaxy.ansible.com/ for more information (you can also browse the available roles there).

If you follow me regarding directory structure your Ansible directory should contain the following structure for now (also see Ansible directory layout best practice):

```
.
├── group_vars
│   ├── all.yml
│   └── k8s.yml
├── hosts
├── host_vars
│   └── k8s-controller1.your-domain.tld
│   └── k8s-controller2.your-domain.tld
│   └── k8s-controller3.your-domain.tld
│   └── localhost
│   └── k8s-worker1.your-domain.tld
│   └── k8s-worker2.your-domain.tld
├── k8s.yml
├── ansible-kubernetes-playbooks
│   └── kubernetes-misc
│       ├── kubeauthconfig
│       ├── kubectlconfig
│       ├── kubedns
│       ├── kubeencryptionconfig
│       ├── kube-router
│       ├── traefik
│       ├── LICENSE
│       └── README.md
└── roles
    ├── brianshumate.consul
    ├── githubixx.cfssl
    ├── githubixx.docker
    ├── githubixx.etcd
    ├── githubixx.harden-linux
    ├── githubixx.kubectl
    ├── githubixx.kubernetes-ca
    ├── githubixx.kubernetes-controller
    ├── githubixx.kubernetes-flanneld
    ├── githubixx.kubernetes-worker
    └── githubixx.peervpn
```

As you can see from the output `group_vars`, `host_vars`, `ansible-kubernetes-playbooks` and `roles` are directories. `hosts` and `k8s.yml` are files. I'll explain what this directories and files are good for while you walk through the blog posts and I'll also tell a little bit more about Ansible.

`Hint`: Quite a few variables are needed by more then one role and playbooks. Put this kind of variables into `group_vars/all.yml`. Especially variables needed by the playbooks (not the roles) fit good there. But it's up to you where you want to place the variables as long as the roles/playbooks find them when they're needed ;-)

That's it for part 1. Continue with part 2 - harden the instances.