Setup Guide

This guide was written as a support for the different Kubernetes courses that I'm teaching. This guide does not intend to show all possible solutions but covers a few options to use Kubernetes. Other options do exist, but I cannot support them in my courses. For questions: mail@sandervanvugt.nl

version 2.1, 22 March 2021: changed weave to calico in AiO procedure, minor changes

Setting up Minikube on a Linux virtual machine

Any virtualization platform will do, but these steps are for VMware Fusion/Workstation of VirtualBox. Installation on MacOS or Windows is also possible, consult the documentation on the Minikube project website for directions. Do consider that while installing directly on top of either Mac or Windows, changes are made to the host OS networking, which may lead to conflicts and which is why I recommend using a dedicated VM instead.

Note: Since nov. 2020 there are problems running nested virtualization in VMware Fusion 12 Pro on Mac OS Big Sur. If you have this configuration, you might want to go for the procedure for Setting up AiO Kubernetes described next.

1. Make sure you have either a fully installed Fedora 31, 33 or Ubuntu 20.04 virtual machine. Other host versions may work but have not been tested yet
    a. 8 GB RAM recommended, 4 GB RAM mimimal
    b. CPU's in the virtual machine have nested virtualization enabled
        i. On VMware: select VM > Processors and Memory > Advanced Options and select both "Enable Hypervisor applications in this virtual machine" and "Enable IOMMU..."
    c. 40 GB disk space recommended, 20 GB minimal
2. Install git, vim and bash completion
    a. On Ubuntu: **sudo apt install git vim bash-completion**
    b. On Fedora: **sudo dnf install git vim bash-completion**
3. As ordinary user with sudo privileges, clone the course Git repository
    a. **git clone https://github.com/sandervanvugt/kubernetes** for Kubernetes in 4 Hours
    b. **git clone https://github.com/sandervanvugt/ckad** for CKAD
    c. **git clone https://github.com/sandervanvugt/microservices** for Microservices
4. Change into the cloned git repo and run the **kube-setup.sh** script
    a. **cd kubernetes**
    b. **./kube-setup**
5. At the end of the script, reboot your virtual machine and run the following:
    a. **minikube start --memory 4096 --vm-driver=kvm2**
6. Once Minikube has started successfully, you'll see a message that it has started. Type **kubectl get all** to verify the minikube cluster is up and running

Setting up AiO-Kubernetes

This method is less common but offers the advantage that no nested virtualization is required. Also it gives nice insight in the workings of Kubernetes.

Notice that this procedure currently only is supported on CentOS 7 with Docker. That does not mean that it doesn't work on Ubuntu, it just means that I haven't had time yet to figure it all out and give full support :-)

1. Install a Centos 7 (NOT 8); minimal installation will do. Theoretically, this works on Ubuntu as well but I haven't had the time yet to make my scripts Ubuntu compatible - hopefully soon.
    a. 20 GB disk space
    b. 4 GB RAM recommended (2 GB minimal)
    c. 2 CPU's
    d. No Swap
2. Install some packages
    a. On CentOS: **yum install git vim bash-comletion**
3. As ordinary user with sudo privileges, clone the course Git repository
    a. **git clone https://github.com/sandervanvugt/kubernetes** for Kubernetes in 4 Hours
    b. **git clone https://github.com/sandervanvugt/ckad** for CKAD
    c. **git clone https://github.com/sandervanvugt/microservices** for Microservices
4. Run the setup scripts using root privileges:
    a. **cd /kubernetes** (or whichever GitHub repository you have cloned)
    b. **./setup-docker.sh**
    c. **./setup-kubetools.sh**
5. In a root shell, install a Kubernetes master node
    a. **kubeadm init --pod-network-cidr=10.10.0.0/16**
6. In a user shell, set up the kubectl client:
    a. **mkdir -p $HOME/.kube**
    b. **sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**
    c. **sudo chown $(id -un):$(id -un) .kube/config**
7. In a user shell, set up the Calico networking agent
    a. **kubectl create -f https://docs.projectcalico.org/manifests/tigera-operator.yaml**
    b. **wget https://docs.projectcalico.org/manifests/custom-resources.yaml**
    c. You now need to define the Pod network, which by default is set to 192.168.0.0/24, which in general is a bad idea. I suggest setting it to 10.10.0.0 - make sure this address range is not yet used for something else!
    d. **sed -i -e s/192.168.0.0/10.10.0.0/g custom-resources.yaml**
    e. **kubectl create -f custom-resources.yaml**
    f. **kubectl get pods -n calico-system**: wait until all pods show a state of Ready, this can take about 5 minutes!
8. By default, user Pods cannot run on the Kubernetes control node. Use the following command to remove the taint so that you can schedule nodes on it:
    kubectl taint nodes --all node-role.kubernetes.io/master-

9. Type **kubectl get all** to verify the cluster works.

Using Hosted Kubernetes in GCE

Kubernetes is supported by many cloud providers. As I cannot be all-inclusive, but most of all, as Google has donated Kubernetes to the world, I like to do something back and explain the procedure on GCE only.

1. From a browser, go to **https://cloud.google.com** and click **Sign in**.
2. Click **Go to console**
3. Select **Compute > Kubernetes Engine > Clusters**
4. Click **Create Cluster**
5. Click **My first cluster**
6. Click **Create Now** and wait a few minutes - typically not more than 5.
7. Once the cluster appears, click **Connect**
8. You'll see a command **gcloud container ...** Click **Run in Cloud Shell** to run it. This will deploy a cloud shell machine, which may take one or two minutes.
   (NOTE: If you don't see the command, this is what you need to type in a cloud shell: **gcloud containers cluster get-credentials cluster-name --zone us-central1-c --project your-project-123**. Make sure the change the cluster name, project name as well as the zone to what applies to your environment.
9. Press Enter to run the command, and click **Authorize** to authorize the cloud shell
10. Type **kubectl get all** to check that the Kubernetes cluster works.