# Practice M2: Container Orchestration

During this practice we will assume that we are working in Linux environment. It could be a physical machine or a virtual one. The distribution of choice is not that important, but it will be better to stick to some of the well supported distributions. You must have Vagrant and VirtualBox or another virtualization solution installed.

All steps can be executed in Windows and/or macOS environment as well.

For the tasks in Part 2, you can download a VM template from this URL: <https://goo.gl/6gyRk7>

## Part 1: Minikube installation

#### Install kubectl

* Download the latest version:

curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl

* Make the file executable:

chmod +x ./kubectl

* Move the binary to our **PATH**:

sudo mv ./kubectl /usr/local/bin/kubectl

#### Shell autocompletion for kubectl

* If running **CentOS**, install **bash-completion** package:

sudo yum install -y bash-completion

* Add autocompletion in the current shell:

source <(kubectl completion bash)

* Add autocompletion to your profile:

echo "source <(kubectl completion bash)" >> ~/.bashrc

#### Install Minikube

* Check which is the latest version here: <https://github.com/kubernetes/minikube/releases>
* Then execute:

curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.31.0/minikube-linux-amd64

* Make the file executable:

chmod +x minikube

* Move the binary to our **PATH**:

sudo mv ./minikube /usr/local/bin/minikube

#### Shell autocompletion for minikube

* If running **CentOS**, install **bash-completion** package:

sudo yum install -y bash-completion

* Add autocompletion in the current shell:

source <(minikube completion bash)

* Add autocompletion to your profile:

echo "source <(minikube completion bash)" >> ~/.bashrc

#### Basic interaction with minikube

* In order to see all supported commands just type:

minikube

* Now that we have **minicube** installed and working, we can start the instance:

minikube start

* Alternatively, we can choose a specific version with:

minikube start –kubernetes-version="v1.13.0"

* We can check the status with:

minikube status

* Or create **ssh** session to the instance:

minikube ssh

* Once inside the instance, we can check our user’s membership:

id

* We can execute few **docker** related commands:

docker version

docker image ls

docker container ps

* Or few **rkt** commands:

rkt version

rkt image list

rkt list

* Let’s close the session:

exit

#### Getting to know kubectl

* Get the list of all available commands:

kubectl

* Now let’s see some information about the cluster:

kubectl cluster-info

* We can check the nodes in the cluster:

kubectl get nodes

* Then get some information about the running pods:

kubectl get pods

* And finally, list all pods including the system ones:

kubectl get pods --all-namespaces

#### Getting to know the dashboard

* Now, before we move on, let’s check the web-based dashboard:

minikube dashboard

* Let’s create a pod visually. We can use the shekeriev/softunit image
* Let’s examine what we have done
* Now we can remove it

#### Working on the command line

In order to save some time, we can use the files provided with the DOF-M2.zip archive. We can extract the files somewhere and enter the folder, there we should find several YAML files.

* Let’s examine the firs file:

vi softunit-pod.yml

* Now we can create a pod out of it:

kubectl create -f softunit-pod.yml

* Once the pod is created, we can check how the Kubernetes sees it:

kubectl describe pod softunit-pod

* Okay, let’s extend our pod. For this, we can use another file, but first let’s compare both files:

vimdiff softunit-pod.yml softunit-pod-ext.yml

* We are ready to apply the changes:

kubectl apply -f softunit-pod-ext.yml

* And check again what has changed for Kubernetes:

kubectl describe pod softunit-pod

* In order to be able to interact with the pod, we must create a service (in imperative way):

kubectl expose pod softunit-pod --name=softunit-svc --target-port=80 --type=NodePort

* If we want to see a brief information about the service we just created, we must execute:

kubectl get svc softunit-svc

* The command for a more detailed information is:

kubectl describe svc softunit-svc

* But what, if we want to see the result? Then we must execute this:

minikube service softunit-svc

* Now let’s remove the service:

kubectl delete svc softunit-svc

* This time we will create the same service, but following the declarative approach:

vi softunit-svc.yml

kubectl create -f softunit-svc.yml

* Again, we can examine the service:

kubectl describe svc softunit-svc

* And then we will remove the pod, but will leave the service, for now:

kubectl delete pod softunit-pod

* Now, it is time put Replication Controllers in use. Let’s examine the file:

vi softunit-rc.yml

* And create the controller:

kubectl create -f softunit-rc.yml

* Are there any pods created?

kubectl get pods

* Did the services registered the new pod’s creation?

kubectl describe svc softunit-svc

* Let’s check the End Points:

kubectl get ep softunit-svc

* Now use a different (JSON) format:

kubectl get ep softunit-svc -o json

* Okay, we don’t need the Replication Controller any more, so let’s delete it:

kubectl delete rc softunit-rc

* It is time to create a deployment, but let’s use an imperative approach:

kubectl run softunit-deploy --image=shekeriev/softunit:v1 --replicas=2 --port=80 --labels=app=softunit

* What if we need to scale our application?

kubectl scale deployment softunit-deploy --replicas=10

* Let’s check the result:

kubectl get pods -w

* And clean up a bit:

kubectl delete deploy softunit-deploy

* Let’s repeat the deployment, but this time in a declarative fashion. Check what is in the file:

vi softunit-deploy-v1.yml

* Create the deployment:

kubectl create -f softunit-deploy-v1.yml

* Inspect the result, first the pods, then the deployment, and then the Replica Set:

kubectl get pod -w

kubectl get deployment -o wide

kubectl describe deployment softunit-deploy

kubectl get rs

kubectl describe rs

* Let’s create a second iteration, but before it, let’s compare both files:

vimdiff softunit-deploy-v1.yml softunit-deploy-v2.yml

* It is time to deploy the new version of our application:

kubectl apply -f softunit-deploy-v2.yml –record

* We can check the deployment status:

kubectl rollout status deployment softunit-deploy

* Once the deployment has ended, we can ask for the history:

kubectl rollout history deployment softunit-deploy

* And roll back to the previous version:

kubectl rollout undo deployment softunit-deploy --to-revision=1

* Again, we can keep an eye on the process with:

kubectl rollout status deployment softunit-deploy

* Once we are done, we can ask for the history again:

kubectl rollout history deployment softunit-deploy

* At the end we can clean up everything

## Part 2: Create a local 3-node cluster

We are going to create a 3-node cluster. Before we begin, we must create one machine with the following parameters:

* 2 CPU
* 2 GB RAM
* 10 GB HDD
* 1 network adapter set to NAT or NAT network
* *1 network adapter set to host only mode*;

These are the minimum required parameters. If you decide to reduce some of them, you may encounter issues during the installation process. Should you go this way, you can try executing the initialization with one additional parameter **--ignore-preflight-errors all**

#### Install the template

* Then we must install the **OS**, which in our case will be **CentOS 7**
* Once ready, we must turn off the **SELinux** both in the current session and permanently with:

sudo setenforce 0

sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

* Then stop the firewall and disable it

sudo systemctl stop firewalld

sudo systemctl disable firewalld

* Do some preparation

sudo yum install -y yum-utils bash-completion

* Add **docker** repository

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

* Add **kubernetes** repository

cat << EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=0

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

* You must check what are the supported / required versions for each component in order to be able to make them work together.
* Finally, execute the following command to install the required software components:

sudo yum install -y docker-ce-18.06.0.ce kubelet kubeadm kubectl kubernetes-cni

* Let’s start the docker and make it start automatically on boot:

sudo systemctl enable docker

sudo systemctl start docker

* Do the same for kubernetes:

sudo systemctl enable kubelet

sudo systemctl start kubelet

* Just in any case change the following system setting:

cat << EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sudo sysctl --system

* One last step – turn off the swap:

sudo swapoff -a

sudo sed -i '/ swap / s/^/#/' /etc/fstab

* Now we are ready to turn off the machine and clone it:

sudo shutdown now

#### Clone VM

* Create two clones
* Do not forget to tick the option to reinitialize the **MAC** address
* Last, but not least, choose full clone

#### Final touches

* Start all three machines
* Set the hostname of all three to **node-X.k8s** where **X** is a number between **1** and **3**
* *Enable the second network adapter and set its* ***IP*** *address to* ***192.168.89.10X****, where* ***X*** *is between* ***1*** *and* ***3***
* Add records for all three hosts in the **/etc/hosts** file

#### Initialize the kubernetes master node

* Initiate the initialization process on the master node:

sudo kubeadm init

* Or if the node has more than one adapter, we can specify the desired one:

sudo kubeadm init --apiserver-advertise-address=192.168.89.101

* Furthermore, we must specify the **CIDR** for pod’s network or other related settings depending on the component we plan to use. We can check the details here:

<https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/#pod-network>

* And for example, if we decide to use the **flannel**, then we should execute the following version of **init**:

sudo kubeadm init --pod-network-cidr 10.244.0.0/16

* Once the initialization process is over, in order to configure our environment, we must execute:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

* Now we can examine our **kubernetes** infrastructure and check the status of the components with:

kubectl get nodes

kubectl get pods -n kube-system

#### Install plugin for the pod network

These two steps - **POD network plugin** installation and **worker nodes** joining can be executed in any order.

* One of the popular plugins is the **Flannel** (<https://github.com/coreos/flannel>). We can install it by executing this on the master:

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

* Once ready, we can check the status of the cluster:

kubectl get node

kubectl get po --all-namespaces

* We can view and edit if needed the configuration of the flannel plugin:

kubectl edit cm -n kube-system kube-flannel-cfg

#### Join worker nodes

* On every worker node execute the generated **join** command:

sudo kubeadm join [IP]:6443 --token [TOKEN] --discovery-token-ca-cert-hash sha256:[HASH]

* If we don’t have the [TOKEN], we can execute this to retrieve it:

kubeadm token list

* Or this one to create new token:

kubeadm token create

* If don’t have the [HASH], then we can retrieve it with:

openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | \

openssl dgst -sha256 -hex | sed 's/^.\* //'

* Once ready, we can check the status of the cluster by executing on the master this:

kubectl get nodes

* If all nodes are marked with status **Ready** we can continue with next steps or get some additional information (by executing the following commands on the master) about the cluster:

kubectl cluster-info

* Or ask for details either for all nodes or for a node:

kubectl describe nodes

kubectl describe nodes node2

* If we want to control the cluster from elsewhere, we can copy the configuration from the master node:

scp root@<master ip>:/etc/kubernetes/admin.conf .

* And then use it as a parameter, for example:

kubectl --kubeconfig ./admin.conf get nodes

#### Dashboard installation and configuration

Like the pod network, the dashboard (<https://github.com/kubernetes/dashboard>) is additional component that can be installed as an extra step.

* In order to install the dashboard, we must execute:

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml

* Now we can get information about all pods running on the cluster:

kubectl describe pods

* Because there are not any other pods beside the system ones, we must modify the command to:

kubectl describe pods --all-namespaces

* If we want information about one pod, for example the dashboard:

kubectl describe pod -l 'k8s-app=kubernetes-dashboard' -n kube-system

* In order to access the dashboard we must execute:

kubectl proxy

* Depending on our setup we could want the dashboard to work on a particular interface:

kubectl proxy --address='10.0.2.15'

* If we visit the specified url we would be surprised that we don’t have credentials. This can be worked out by creating a file dashboard-admin-user.yml:

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

* And add the resource by:

kubectl apply -f dashboard-admin-user.yml

* Then create additional one dashboard-admin-role.yml:

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kube-system

* And add the resource by:

kubectl apply -f dashboard-admin-role.yml

* Then we can get information about the secret of the admin user:

kubectl -n kube-system get secret | grep admin-user | awk '{print $1}'

* Or get the secret itself:

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

* Now we can copy the token and use it to login to the dashboard
* Once logged in successfully we can examine the dashboard
* If we want to connect to the API Server from elsewhere, we can copy the configuration from the master node:

scp root@<master ip>:/etc/kubernetes/admin.conf .

* And then use it as a parameter to the proxy command:

kubectl --kubeconfig ./admin.conf proxy

* Now we can use the following url:

<http://localhost:8001/api/v1>

#### Clean up resources in case of error or start from the beginning

Unfortunately, not everything goes always the way we want it. In such cases, we can use some of the following commands.

* Grep for the status of a pod:

kubectl get po -n kube-system | grep [pod-name]

* Get the logs of a pod:

kubectl logs -n kube-system [pod-name]

* View or edit configuration:

kubectl edit cm -n kube-system [config-name]

* Delete a pod:

kubectl delete po -n kube-system -l [label-name]=[label-value]

* If we want to explore the events:

kubectl get events

* We can drain a node:

kubectl drain [node-name] --delete-local-data --force --ignore-daemonsets

* And delete a node:

kubectl delete node [node-name]

* As a last resort, we can undo any changes mage by **kubeadm init** or **kubeadm join**:

kubeadm reset

* And then start from the beginning with **kubeadm init** or **kubeadm join**

## Part 3: Use Vagrant to create a local 3-node cluster

Now we can put all manual steps together and wrap them in a **Vagrantfile**. This way we can produce a whole **Kubernetes** cluster in an automated fashion.

The **Vagrant** file should contain the following blocks:

* In the beginning of the file put this:

Vagrant.configure(2) do |config|

config.ssh.insert\_key = false

* One for the master node:

config.vm.define "k8s1" do |k8s1|

k8s1.vm.box = "shekeriev/centos-7-64-minimal"

config.vm.provider "virtualbox" do |v|

v.memory = 2048

v.cpus = 2

end

k8s1.vm.hostname = "k8s1.sulab.local"

k8s1.vm.network "private\_network", ip: "192.168.99.101"

k8s1.vm.synced\_folder "vagrant/", "/vagrant"

k8s1.vm.provision "shell", inline: <<EOS

echo "\* Add hosts ..."

echo "192.168.99.101 k8s1.sulab.local k8s1" >> /etc/hosts

echo "192.168.99.102 k8s2.sulab.local k8s2" >> /etc/hosts

echo "192.168.99.103 k8s3.sulab.local k8s3" >> /etc/hosts

echo "\* Stop Firewall ..."

sudo systemctl stop firewalld

sudo systemctl disable firewalld

echo "\* Change SELinux state ..."

sudo setenforce 0

sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

echo "\* Install Prerequisites ..."

sudo yum install -y yum-utils bash-completion

echo "\* Add Docker repository ..."

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

echo "\* Add Kubernetes repository ..."

cat << EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=0

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

echo "\* Install Docker and Kubernetes ..."

sudo yum makecache fast

# sudo yum update -y

sudo yum install -y docker-ce-18.06.0.ce kubelet kubeadm kubectl kubernetes-cni

echo "\* Start Docker ..."

sudo systemctl enable docker

sudo systemctl start docker

echo "\* Start Kubernetes ..."

sudo systemctl enable kubelet

sudo systemctl start kubelet

echo "\* Change some system settings ..."

cat << EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sudo sysctl --system

echo "\* Turn off the swap ..."

sudo swapoff -a

sudo sed -i '/ swap / s/^/#/' /etc/fstab

echo "\* Add vagrant user to docker group ..."

sudo usermod -aG docker vagrant

echo "\* Initialize Kubernetes cluster ..."

sudo kubeadm init --apiserver-advertise-address=192.168.99.101 --pod-network-cidr 10.244.0.0/16

echo "\* Copy configuration for root ..."

mkdir -p /root/.kube

sudo cp -i /etc/kubernetes/admin.conf /root/.kube/config

sudo chown root:root /root/.kube/config

echo "\* Copy configuration for vagrant ..."

mkdir -p /home/vagrant/.kube

sudo cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config

sudo chown vagrant:vagrant /home/vagrant/.kube/config

echo "\* Install POD network plugin ..."

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

echo "\* Install Dashboard ..."

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml

echo "\* Create admin user ..."

cat << EOF > /vagrant/dashboard-admin-user.yml

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

EOF

echo "\* Create role ..."

cat << EOF > /vagrant/dashboard-admin-role.yml

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kube-system

EOF

echo "\* Add both the user and role ..."

kubectl apply -f /vagrant/dashboard-admin-user.yml

kubectl apply -f /vagrant/dashboard-admin-role.yml

echo "\* Save the user token ..."

kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}') > /vagrant/admin-user-token.txt

echo "\* Create custom token ..."

kubeadm token create abcdef.1234567890abcdef

echo "\* Save the hash to a file ..."

openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.\* //' > /vagrant/hash.txt

EOS

end

* One for worker node #1

config.vm.define "k8s2" do |k8s2|

k8s2.vm.box = "shekeriev/centos-7-64-minimal"

config.vm.provider "virtualbox" do |v|

v.memory = 2048

v.cpus = 2

end

k8s2.vm.hostname = "k8s2.sulab.local"

k8s2.vm.network "private\_network", ip: "192.168.99.102"

k8s2.vm.synced\_folder "vagrant/", "/vagrant"

k8s2.vm.provision "shell", inline: <<EOS

echo "\* Add hosts ..."

echo "192.168.99.101 k8s1.sulab.local k8s1" >> /etc/hosts

echo "192.168.99.102 k8s2.sulab.local k8s2" >> /etc/hosts

echo "192.168.99.103 k8s3.sulab.local k8s3" >> /etc/hosts

echo "\* Stop Firewall ..."

sudo systemctl stop firewalld

sudo systemctl disable firewalld

echo "\* Change SELinux state ..."

sudo setenforce 0

sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

echo "\* Install Prerequisites ..."

sudo yum install -y yum-utils bash-completion

echo "\* Add Docker repository ..."

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

echo "\* Add Kubernetes repository ..."

cat << EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=0

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

echo "\* Install Docker and Kubernetes ..."

sudo yum makecache fast

#sudo yum update -y

sudo yum install -y docker-ce-18.06.0.ce kubelet kubeadm kubectl kubernetes-cni

echo "\* Start Docker ..."

sudo systemctl enable docker

sudo systemctl start docker

echo "\* Start Kubernetes ..."

sudo systemctl enable kubelet

sudo systemctl start kubelet

echo "\* Change some system settings ..."

cat << EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sudo sysctl --system

echo "\* Turn off the swap ..."

sudo swapoff -a

sudo sed -i '/ swap / s/^/#/' /etc/fstab

echo "\* Add vagrant user to docker group ..."

sudo usermod -aG docker vagrant

echo "\* Join the worker node (k8s2) ..."

sudo kubeadm join 192.168.99.101:6443 --token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:`cat /vagrant/hash.txt`

EOS

end

* And one for the worker node #2 use the same as for node #1, but adjust the corresponding values
* Don’t forget to put the closing instruction:

end

* Now we are ready to save the file and initiate the creation of the cluster:

time vagrant up

* We can logon to the master and experiment with some of tasks we did in the previous part
* And finally, we can clean up everything by executing:

vagrant destroy --force