

# Creating Kubernetes cluster with k3s on multipass

---

 [gdngn.com/deploying-k3s-with-multipass-and-cloud-init](https://gdngn.com/deploying-k3s-with-multipass-and-cloud-init)

22 January 2023

By [Adnan Selimovic](#) in [Experience](#) — Jan 22, 2023 — Takes 7 minutes

Setting up a Kubernetes lab is often a necessity in my daily work. Testing various software, understanding how to run it, configure it, learning new features, and so on. It's part of the job.

Running directly on the cloud could raise costs and using local development tools is a choice I am always going with (when possible).

In my experience I've found that [k3s](#) is a suitable tool for the job:

- It's easy to install
- It's easy to configure
- Requires low resources

Since working on different machines and operating systems I've standardized the setup to work only on Ubuntu. [Multipass](#) is a tool that comes to the rescue. It allows me to create virtual machines from the terminal and makes my life easier.

Another neat feature is utilizing [cloud-init](#) for the provisioning of VM instances on creation.

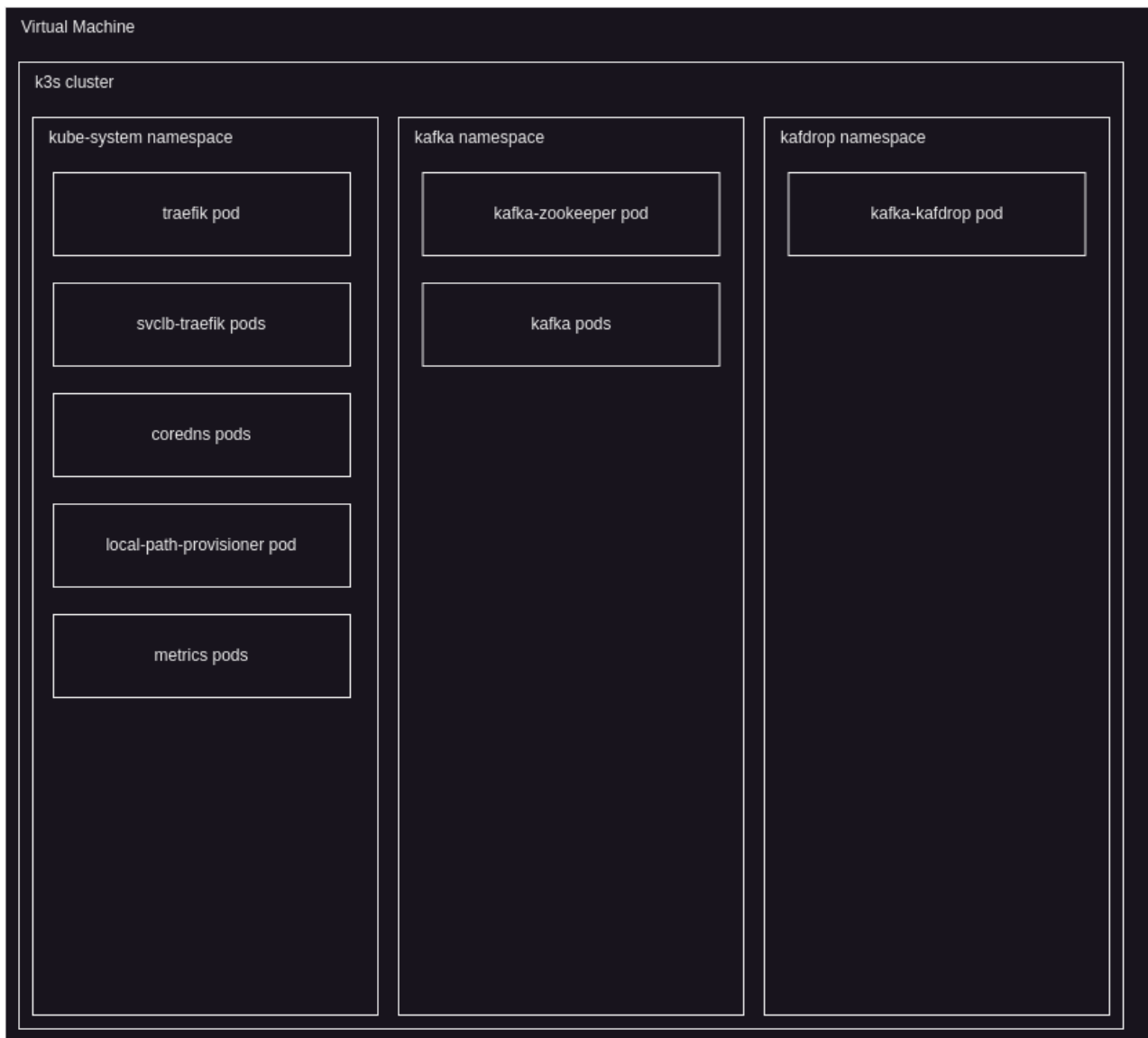
So my recipe for the homemade Kubernetes laboratory is:

- K3s
- Multipass
- cloud-init

## Architecture

---

The resulting Kubernetes cluster will look like the one in the picture below.



Resulting Kubernetes cluster.

What you get in the end is a Kubernetes cluster accessible from the host operating system.

→ ~ multipass list

Name	State	IPv4	Image
k3s	Running	10.117.145.168 172.17.0.1 10.42.0.0 10.42.0.1	Ubuntu 22.04 LTS

→ ~ hostname

qdnqn

→ ~ curl 10.117.145.168

404 page not found

→ ~ curl 10.117.145.168/kafdrop -IL

HTTP/1.1 302 Found

Content-Length: 0

Date: Sun, 22 Jan 2023 13:24:45 GMT

Location: http://10.117.145.168/kafdrop/

HTTP/1.1 406 Not Acceptable

Access-Control-Allow-Credentials: true

Access-Control-Allow-Headers: Origin,Accept,X-Requested-With,Content-Type,Access-Control-Request-Method,Access-Control-Request-Headers,Authorization

Access-Control-Allow-Methods: GET,POST,PUT,DELETE

Access-Control-Allow-Origin: \*

Access-Control-Max-Age: 3600

Content-Language: en-US

Content-Length: 10807

Content-Type: text/html; charset=UTF-8

Date: Sun, 22 Jan 2023 13:24:46 GMT

This is made possible by the svc-lb-traefik pod. If you are interested in the configuration of the Traefik on k3s check the blog post below.

As you can see in the picture below, the process is broken into a few steps:

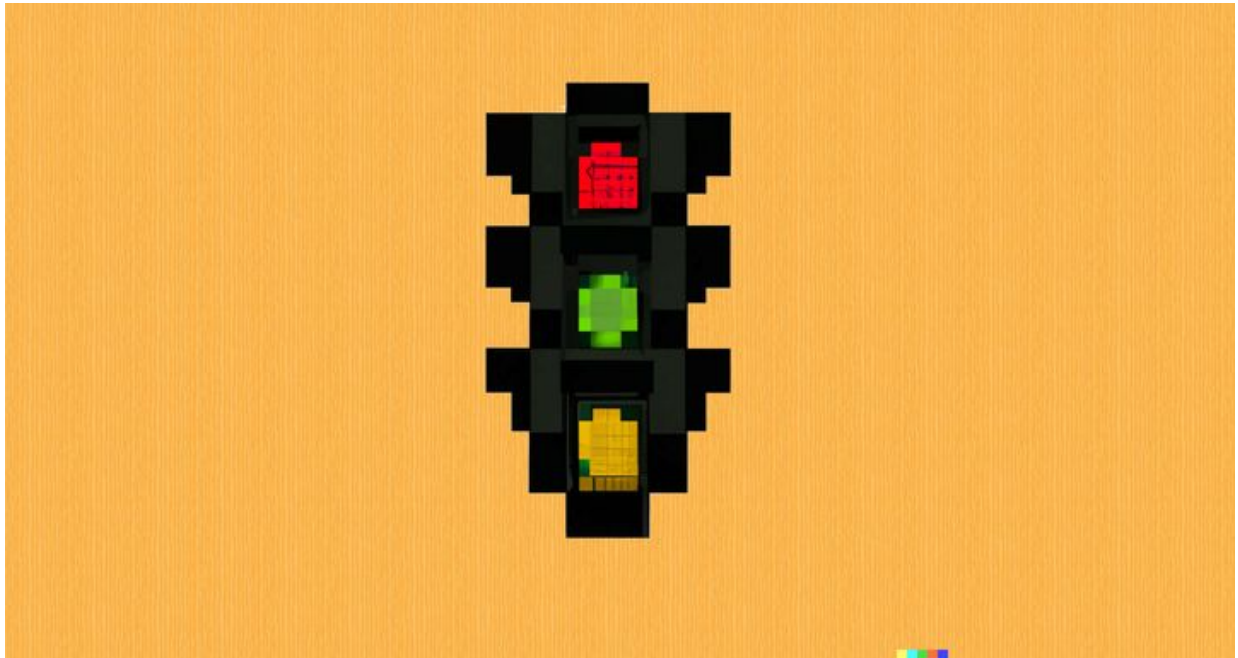
- Launch multipass command for creating VM
- Provision the new VM using cloud-init
- Run commands on the first boot using cloud-init
- Run setup after the provision is completed using .bash\_profile (running only once on the first boot)
- Done

## Traffic engineering with Traefik on k3s distribution of Kubernetes

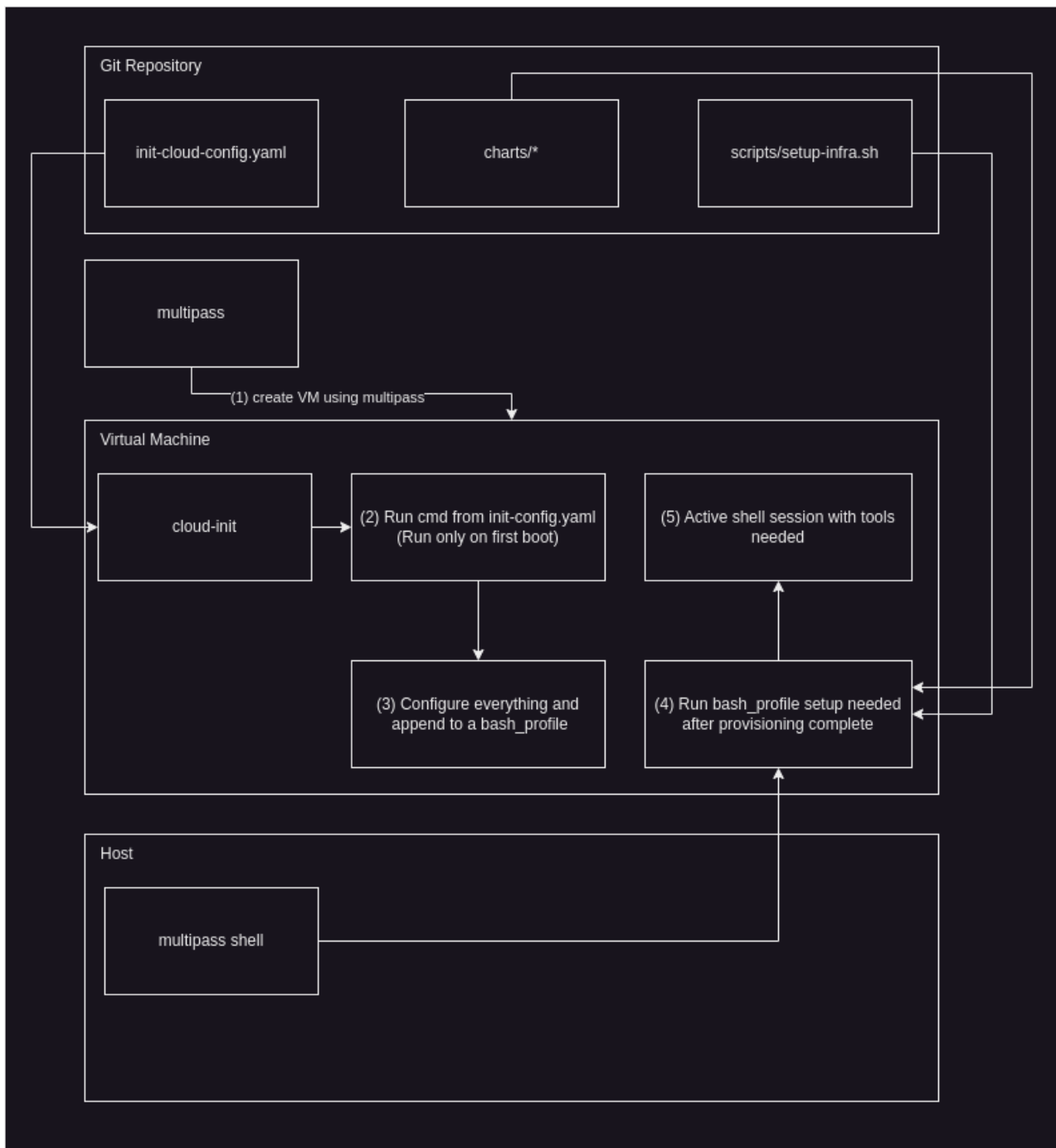
Traefik is one of the most popular ingress controllers on Kubernetes. Traefik v2 brought some major changes in the usage of the controller itself. It brought the approach of heavy usage of Custom Resources on Kubernetes to provide reconfigurability and expanded fields of operation apart from the Ing...



qdnqnAdnan Selimovic



You also might be interested in this article.



Process of setting up and provisioning the Kubernetes cluster on the Virtual machine.

## Multipass

Launching a new machine using multipass is given in the `/.start.sh` script. Let's look at the contents of it.

```
#!/bin/bash
```

```
cd "$(dirname "$0")"
echo "$(pwd) is current working directory."
```

```
multipass launch -n k3s --mem 4G --disk 40G --cpus 4 --cloud-init init-config.yaml
--mount ./:/home/ubuntu/cloud-native-infrastructure
```

Changing the current working directory to the directory of the location of the script. After that multipass is starting the VM.

The cloud-init configuration is given in the `--cloud-init init-config.yaml`. The file `/init-config.yaml` holds all the information for the cloud-init provisioning process.

After the cloud-init provision process is done multipass will mount this repository in the home directory of the ubuntu user.

Let's review `init-config.yaml`. For example, Run commands have a duty to install needed software for the VM.

```
runcmd:
  - 'curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"'
  - 'curl -LO "https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"'
  - 'echo "$(cat kubectl.sha256) kubectl" | sha256sum --check'
  - 'install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl'
  - 'mkdir -p /etc/apt/keyrings'
  - 'curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /etc/apt/keyrings/docker.gpg'
  - 'echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null'
  - 'apt-get update'
  - 'apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin -
y'
  - 'snap install go --classic'
  - 'curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3'
  - 'chmod 700 get_helm.sh'
  - './get_helm.sh'
  - 'wget https://go.dev/dl/go1.19.linux-amd64.tar.gz'
  - 'sudo tar -xvf go1.19.linux-amd64.tar.gz'
  - 'sudo mv go /usr/local'
  - 'echo "$(hostname -I | cut -d" " -f1) k3s.local.registry.com" >> /etc/hosts'
  - 'curl -sFL https://get.k3s.io | sh -'
  - 'echo 'eval `ssh-agent`' >> /etc/bash.bashrc'
  - 'echo "sudo chown -R ubuntu /etc/rancher" >> /etc/bash.bashrc'
  - 'echo "sudo chown ubuntu /var/run/docker.sock" >> /etc/bash.bashrc'
  - 'echo "export KUBECONFIG=/etc/rancher/k3s/k3s.yaml" >> /etc/bash.bashrc'
  - 'echo "alias k=kubectl" >> /etc/bash.bashrc'
  - 'ssh-keyscan -t rsa github.com >> /etc/ssh/ssh_known_hosts'
  - 'eval `ssh-agent`'
  - 'cd /home/ubuntu/'
  - 'chown -R ubuntu /home/ubuntu'
  - 'touch /home/ubuntu/.cloud-init.done'
```

This is the part where you want to add software before the actual login. If you want to add any commands to run before the first login - add them here.

I am using this to install kubectl, helm, k3s, go, docker, change ownership in the VM to the ubuntu user (default user on k3s), make an alias for kubectl, and create a file when cloud-init is done: `.cloud-init.done` .

The file `.cloud-init.done` is used by `~/.bash_profile` to know when the cloud-init process of provisioning is done.

The `~/.bash_profile` is appended using the `write_files` method from cloud-init.

```

write_files:
- content: |
    {
      "insecure-registries" : ["k3s.local.registry.com:5000"]
    }
  path: /etc/docker/daemon.json
- content: |
    mirrors:
      k3s.local.registry.com:5000:
        endpoint:
          - "http://k3s.local.registry.com:5000"
  path: /etc/rancher/k3s/registries.yaml
  append: true
- content: |
    #!/bin/bash
    SP="/-\\|"
    if [[ ! -f "/home/ubuntu/.cloud-init.done" ]]; then
      echo -n "Waiting for cloud init to finish "
    fi

    while [[ ! -f "/home/ubuntu/.cloud-init.done" ]];
    do
      printf "\b${SP:i++%${#SP}:1}"
      sleep 1
    done;

    if [[ ! -d "/home/ubuntu/cloud-native-infrastructure" ]]; then
      echo -n "Waiting for mount of directory "
    fi

    while [[ ! -d "/home/ubuntu/cloud-native-infrastructure" ]];
    do
      printf "\b${SP:i++%${#SP}:1}"
      sleep 1
    done;

    eval `ssh-agent`
    sudo chown ubuntu /var/run/docker.sock

    cd /home/ubuntu/

    cd /home/ubuntu/cloud-native-infrastructure
    ./scripts/setup-infrastructure.sh

    export GOROOT=/usr/local/go
    export PATH=$GOROOT/bin:$PATH
  path: /home/ubuntu/.bash_profile

```

As can be seen when spawning a shell this shell script will also run and do the next things:

- Wait for the cloud-init provision process to complete using a semaphore file `.cloud-init.done`
- Change ownership of the docker socket to the current user
- Run the script from the repository `/scripts/setup-infrastructure.sh`
- Add Go to the path



The next important part is the `scripts/setup-infrastructure.sh`.

This script holds the next content.

```
#!/bin/bash

cd "$(dirname "$0")"

if [[ ! -f "/home/qdnqn/.run.once" ]]; then
    export KUBECONFIG=/etc/rancher/k3s/k3s.yaml

    while [[ $(kubectl get crd | grep ingressroutes.traefik.containo.us | wc -l) == 0 ]];
    do
        sleep 5
    done;

    docker run -d -p 5000:5000 --restart=always --name registry registry:2

    kubectl create ns kafka
    kubectl create ns kafdrop

    kubectl apply -f resources/raw/yaml/setup/traefik-config-k3s.yaml

    helm upgrade --install kafka ../charts/kafka --namespace kafka --values
    ../charts/kafka/values.yaml
    helm upgrade --install kafdrop ../charts/kafdrop --namespace kafdrop --values
    ../charts/kafdrop/values.yaml

    VM_IP=$(hostname -I | cut -d " " -f1)
    sed -i "s/{VM_IP}/{VM_IP}/g" resources/raw/yaml/setup/ingresses.yaml

    kubectl apply -f resources/raw/yaml/setup/ingresses.yaml

    touch /home/ubuntu/.run.once
fi
```

This script will run only once. Using the same semaphore file technique only this file is named `.run.once`.



Multipass and cloud-init currently don't support any way of notification when the provisioning process is done. Hence using the semaphore files technique is my current hack around this issue. I don't want the script `./setup-infrastructure.sh` to run before the cloud-init is done and mounting of the repository inside the VM is done.

At the first spawn of the shell inside the VM using `multipass shell k3s`, the setup-infrastructure script will start the docker registry inside the VM but outside of the Kubernetes cluster.

It will create needed namespaces and ingresses from the resources dir.

It will `helm upgrade --install` needed charts.



You can modify this script to your needs. The example is given for Kafka and Kafdrop. You can add as many charts as you need and modify the script to do what you need on the startup.

At the end of the script, it will create `.run.once` file in the home directory and this part will not be run at the subsequent spawns of the shell.

---

The process itself can take around 5-10 minutes to complete. Depends on your machine and internet speed but you will get a fully functional Kubernetes cluster.

After reading the post I suggest reviewing the architecture image of the process itself.

If you want to get the repository, navigate to the link below.

[GitHub - qdnqn/cloud-native-infrastructure: Infrastructure for running cloud-native-go.](#)

[Infrastructure for running cloud-native-go. Contribute to qdnqn/cloud-native-infrastructure development by creating an account on GitHub.](#)



[GitHubqdnqn](#)

# qdnqn/cloud-native-infrastructure

Infrastructure for running cloud-native-go.



 1 Contributor  
 0 Issues  
 0 Stars  
 0 Forks



---


If you need any help or suggest improvements please leave a comment to start a conversation.

Thank you for subscribing and readin

## Subscribe to qdnqn

---

Don't miss out on the latest issues. Sign up now to get access to the library of members-only issues.

 [qdnqn@example.com](mailto:qdnqn@example.com)  
[Subscribe](#)

Join other 10 members. Unsubscribe whenever you want.