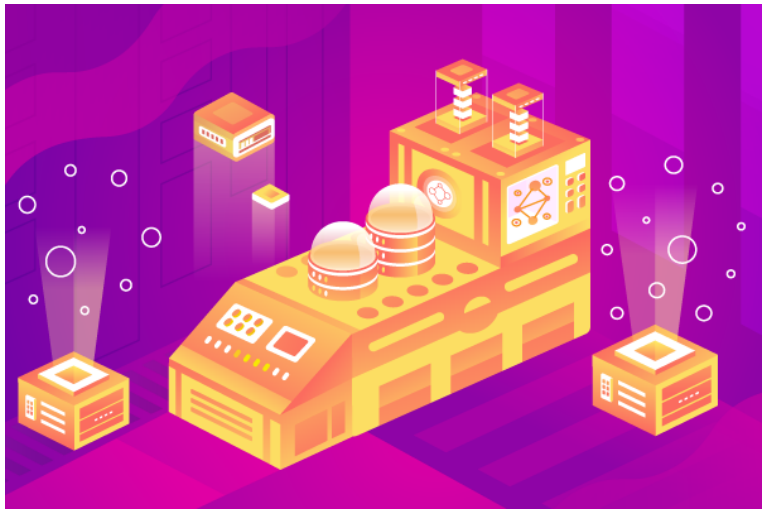

Introduction to K3s

Study Guide

Will Boyd
willb@linuxacademy.com
June 10, 2020

Contents

Introducing K3s

Documentation:

- **K3s - Lightweight Kubernetes**

K3s is a simple, lightweight Kubernetes distribution. It is designed for situations where compute resources may be limited, or where a simple and easy Kubernetes solution is needed.

Advantages of K3s:

- Lightweight:

  K3s is designed to use half the memory of a regular K8s installation and uses a single <100MB binary.
- Simple:

  K3s packages dependencies and add-ons within a single binary.
- Easy:

  K3s provides sensible, secure defaults and automates complex cluster operations like distributing certificates.

K3s Background:

- Who made it?

  K3s was built by the kind folks at Rancher.
- Why is it called K3s?

  Kubernetes is 10 letters and is stylized as K8s (the 8 represents 8 letters). A similar 5-letter word would be half the size of Kubernetes, with 3 letters between the K and the S.

- How do you pronounce it?

  K3s has no official pronunciation.

Use Cases:

- Edge

  Edge devices often have resource constraints. K3s is lightweight and can help here. Its simplicity also makes updates easier to manage.

- Internet of Things (IoT)

  IoT devices can also be resource-constrained and difficult to update.

- Continuous Integration (CI)

  Simple and lightweight, it can be an easy addition to your CI processes.

- Development

  Need a quick and easy cluster for development purposes?

- ARM or Embedded

  Like Edge and IoT devices, ARM and embedded devices often impose resource constraints and limitations on updates.

- Anytime you need simple K8s

  Whenever you need a simple and hassle-free Kubernetes cluster, K3s can help!

Exploring How K3s Works

Documentation:

- **K3s - How it Works**

Technical Features:

- Lightweight

  K3s uses half the memory of regular Kubernetes.

- Single Binary

  A single <100MB binary contains all Kubernetes control plane components. K3s also bundles a containered runtime and other dependencies.

- sqlite3 Stroage Backend

  K3s uses a lightweight sqlite-based storage backend instead of the usual `etcd`. This is also included in the K3s binary.

- Packaged Add-ons

  K3s includes a few packaged add-ons to provide out-of-the-box functionality, such as flannel CNI, a Helm controller, and Traefik ingress controller.

- Secure-by-Default

  K3s includes sensible defaults for security, and internally manages certificate distribution between components.

Installing K3s

Documentation:

- **K3s Quick Start**
- **Installation Requirements**

System Requirements:

- Minimum 512 MB Ram
- Minimum 1 CPU
- SSD (Solid State Disk) Recommended
- Operating System (OS):
    K3s is officially tested on the following operating systems:
        Ubuntu 16.04
        Ubuntu 18.04
        Raspian Buster
- Ports:
    API Server — All nodes need to be able to reach the K3s server on port 6443 via TCP.
    Flannel VXLAN — All server and agent nodes need to be able to reach each other on port 8472 via UDP.
    Metrics Server — If you want to use metrics server, open 10250 on all server and agent nodes via TCP.

You can install K3s using the K3s installation script located at **https://get.k3s.io**.

```
curl -sfL https://get.k3s.io | sh -
```

Building a K3s Cluster

Documentation:

- **K3s Quick Start**

To install K3s on a worker node, you need to supply two environment variables:

- `K3S_URL` — The URL of your K3s server
- `K3S_TOKEN` — A token used to authenticate with the K3s server

  You can find it at `/var/lib/rancher/k3s/server/node-token` on the K3s server.

```
curl -sfL https://get.k3s.io | K3S_URL=https://$K3S_SERVER_PRIVATE_IP:6443 K3S_TOKEN=$NODE_TOKEN
sh -
```

Running an Application in K3s

If you wish to run `kubectl` commands as a user other than `root`, you will need to grant that user permission to read the K3s kubeconfig file:

```
sudo groupadd k3s

sudo usermod -a -G k3s user

sudo chown root:k3s /etc/rancher/k3s/k3s.yaml

sudo chmod 740 /etc/rancher/k3s/k3s.yaml
```

You can run applications by creating pods with `kubectl`, just like you would in any other Kubernetes cluster.

Create a pod definition file:

```
vi aquarium.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: aquarium
spec:
  containers:
  - name: funbox
    image: wernight/funbox
    command: ["asciiquarium"]
```

Create the pod:

```
kubectl create -f aquarium.yml
```