Five Kubernetes distributions for local environments

This article contains benchmarks and reviews of the following Kubernetes distributions:

* [minikube](https://minikube.sigs.k8s.io/docs/)
* [microk8s](https://microk8s.io/)
* [k3s](https://k3s.io/)
* [k0s](https://k0sproject.io/)
* [kind](https://kind.sigs.k8s.io/)

Minikube:

[Minikube is the oldest and most popular Kubernetes distribution for local environments. The project started in 2016. Its default configuration makes it very easy to get started. Most development tools support minikube out of the box. Minikube supports various “driver”s which allow you to run in different set-ups (virtualbox, docker, bare-metal, etc). For docker driver to work, docker has to be installed on the system and the current user has to have the right permissions.](https://github.com/kubernetes/minikube)

[Minikube](https://github.com/kubernetes/minikube) has the primary goals of being the best tool for local Kubernetes application development, and to support all Kubernetes features that fit. No matter what operating system you use, [Minikube's documentation](https://minikube.sigs.k8s.io/docs) offers an easy [installation](https://minikube.sigs.k8s.io/docs/start/).

Minikube quickly sets up a local Kubernetes cluster on Linux, macOS, or Windows with the following features:

* Supports the latest Kubernetes release (+6 previous minor versions)
* Cross-platform (Linux, macOS, Windows)
* Deploys as a VM, a container, or on bare-metal
* Multiple container runtimes (CRI-O, containerd, Docker)
* Docker API endpoint for blazing-fast image pushes
* LoadBalancer, filesystem mounts, FeatureGates, and other advanced features
* Add-ons for easily installing Kubernetes applications

Because Minikube is an open source project, you can contribute to its [source code](https://github.com/kubernetes/minikube).

### microk8s

MicroK8s is [Canonical](https://canonical.com/)‘s way of getting into Kubernetes market. It’s a single “snap” command to install on Ubuntu systems. Internally MicroK8s uses containers as the container runtime which is bundled into the snap package.

Microk8s is a small, lightweight, and fully conformant Kubernetes distribution from Canonical. It’s a minimalistic distribution focused on simplicity and performance. Given its footprint, Microk8s can be easily deployed in IoT and edge devices.

Canonical has packaged [Microk8s](https://microk8s.io/) as a snap, the company’s package manager for Linux. A snap is a bundle of an app and its dependencies that works without modification across many different Linux distributions. Snaps are independent, self-contained applications that run in a sandbox with mediated access to the host system. Snaps have emerged as an alternative to the standard .deb packages typically used in Debian-based distributions. Applications packaged as a snap can be easily installed and uninstalled.

Apart from the recent versions of Ubuntu, snaps can be deployed on a variety of platforms, including Linux Mint, Raspberry Pi OS, and Arch Linux.

During the benchmark microk8s did the worst. I am not sure what the slowness was caused by but my guess is its resource use.

Storage and DNS addons are disabled by default so they need to be enabled if persistence and DNS. Microk8s is the only distribution that disables DNS by default. During benchmarking I had issues with helm v3 packaged with microk8s because it was an early release which had a [bug](https://github.com/helm/helm/issues/7955).

### K3s

K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances.

k3s is a distribution from [Rancher](https://rancher.com/) and is a [CNCF](https://www.cncf.io/) project. k3s binary contains all the necessary cluster components so the cluster setup is very quick (the quickest of the 5 distributions). Although it comes with both storage and DNS installed, it is also very light. k3s doesn’t have any dependencies.

k3s is very easy to use but as it’s not as widely used external tool support for it is not as good as minikube. It also lacks the shortcuts useful for beginners (e.g. helm works out of the box with minikube, but with k3s you have to extract kubeconfig from kubectl using “k3s kubectl config view –raw”).

K3s is packaged as a single <40MB binary that reduces the dependencies and steps needed to install, run and auto-update a production Kubernetes cluster.

K3s is a fully compliant Kubernetes distribution with the following enhancements:

* Packaged as a single binary.
* Lightweight storage backend based on sqlite3 as the default storage mechanism. etcd3, MySQL, Postgres also still available.
* Wrapped in simple launcher that handles a lot of the complexity of TLS and options.
* Secure by default with reasonable defaults for lightweight environments.
* Simple but powerful “batteries-included” features have been added, such as: a local storage provider, a service load balancer, a Helm controller, and the Traefik ingress controller.
* Operation of all Kubernetes control plane components is encapsulated in a single binary and process. This allows K3s to automate and manage complex cluster operations like distributing certificates.
* External dependencies have been minimized (just a modern kernel and cgroup mounts needed). K3s packages required dependencies, including:
  + containerd
  + Flannel
  + CoreDNS
  + CNI
  + Host utilities (iptables, socat, etc)
  + Ingress controller (traefik)
  + Embedded service loadbalancer
  + Embedded network policy controller

### k0s

k0s is similar to k3s in many ways. Like k3s, k0s also comes as a single binary which achieves a very quick setup times. k0s binary is massive (170MB) compared to k3s (50MB). You can add k0s to startup by using “k0s install controller” command or can run it quickly using “k0s controller” command. k0s, like k3s doesn’t have any dependencies.

Because of it’s low resource footprint I was surprised by the benchmark results. The average response time was similar to the response times achieved with microk8s.

k0s does not come with a storage handler. There isn’t even a command to install it. I had to set up helm and install [rook](https://rook.io/). For that reason it got the worst ease of use score.

k0s drastically reduces the complexity of installing and running a fully conformant Kubernetes distribution. New kube clusters can be bootstrapped in minutes. Developer friction is reduced to zero, allowing anyone, with no special skills or expertise in Kubernetes to easily get started.

k0s is distributed as a single binary with no host OS dependencies besides the host OS kernel. It works with any operating system without additional software packages or configuration. Any security vulnerabilities or perf issues can be fixed directly in the k0s distribution.

k0s is completely free for personal or commercial use, and it always will be. The source code is available on [Github](https://github.com/k0sproject/k0s) under Apache 2 license. It's a no brainer foundation for any Kubernetes projects and easy build upon.

### kind

[kind](https://kind.sigs.k8s.io/)'s developers describe it as "a tool for running local Kubernetes clusters using Docker container 'nodes.'" It was designed for testing Kubernetes but may also be used for local development or continuous integration.

Kind is another Kubernetes SIGs project but is quite different compared to minikube. As the name suggests it moves the cluster into Docker containers. This leads to a significantly faster startup speed compared to spawning VM.

Creating a cluster is very similar to minikube’s approach. Executing kind create cluster, playing the waiting game and afterwards you are good to go. By using different names (--name) kind allows you to create multiple instances in parallel.

One feature that I personally enjoy is the ability to load my local images directly into the cluster. This saves me a few extra steps of setting up a registry and pushing my image each and every time I want to try out my changes. With a simple kind load docker-image my-app:latest the image is available for use in my cluster. Very nice!

If you are looking for a way to programmatically create a Kubernetes cluster, kind kindly (you have been for waiting for this, don’t you :P) publishes its Go packages that are used under the hood. If you want to get to know more have a look at the GoDocs and check out how KUDO uses kind for their integration tests.

kind supports:

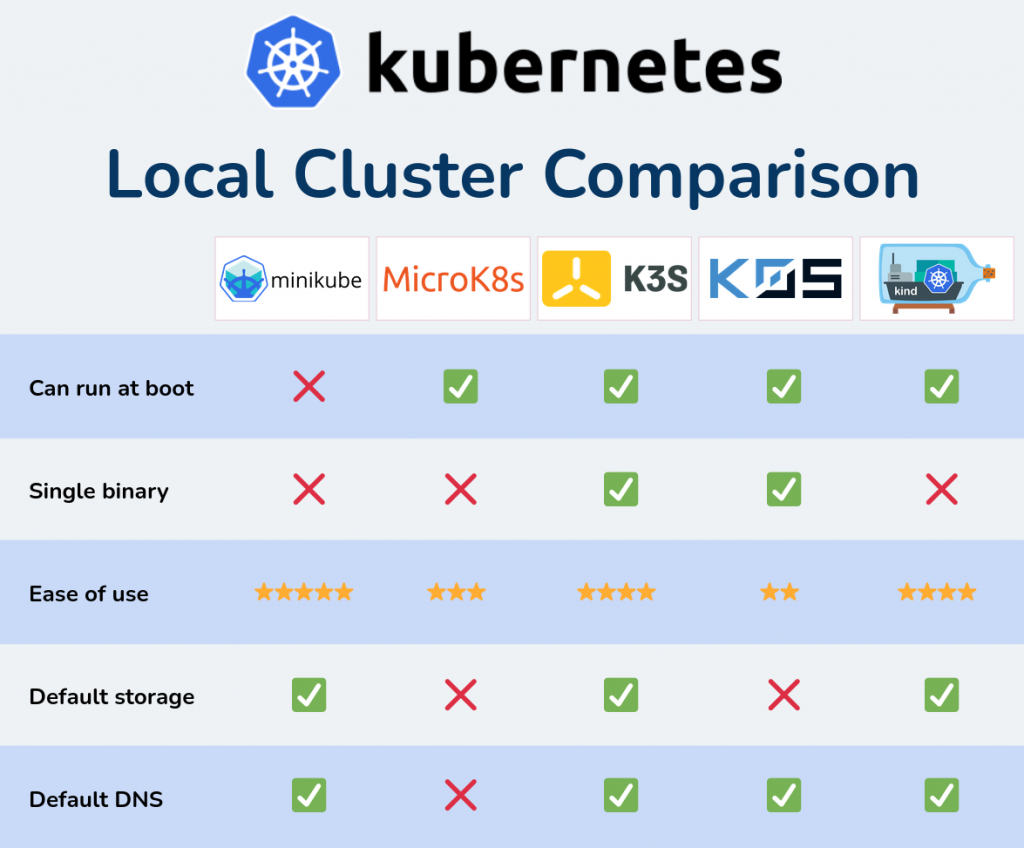
* Multi-node (including high-availability) clusters
* Building Kubernetes release builds from source
* Make/Bash/Docker or Bazel, in addition to pre-published builds
* Linux, macOS, and Windows

In addition, kind is a Cloud Native Computing Foundation (CNCF)-certified conformant Kubernetes installer. Because it's open source, you can find kind's [source code](https://github.com/kubernetes-sigs/kind) in its GitHub repository.

## Feature comparison

When comparing distributions I made the following assumptions:

* minikube can actually run at but only if you define the service manually. k0s has a command to define the service. k3s will run at boot if installed by following the [quick start guide](https://rancher.com/docs/k3s/latest/en/quick-start/)
* Subtracted ease of use points for each obstacle encountered during the installation of wordpress helm chart (missing addon, no default kubeconfig)
* Considered k3s and k0s as single binary because they have no dependencies and do no downloads during the set up process.



## Verdict

After spending hours testing and benchmarking these tools I’ll probably continue using minikube because it’s so easy to use. I will keep watching kind and k3s to see if they get better. Speed of k3s is its best advantage and I’m sure that alone will make lots of people use it.