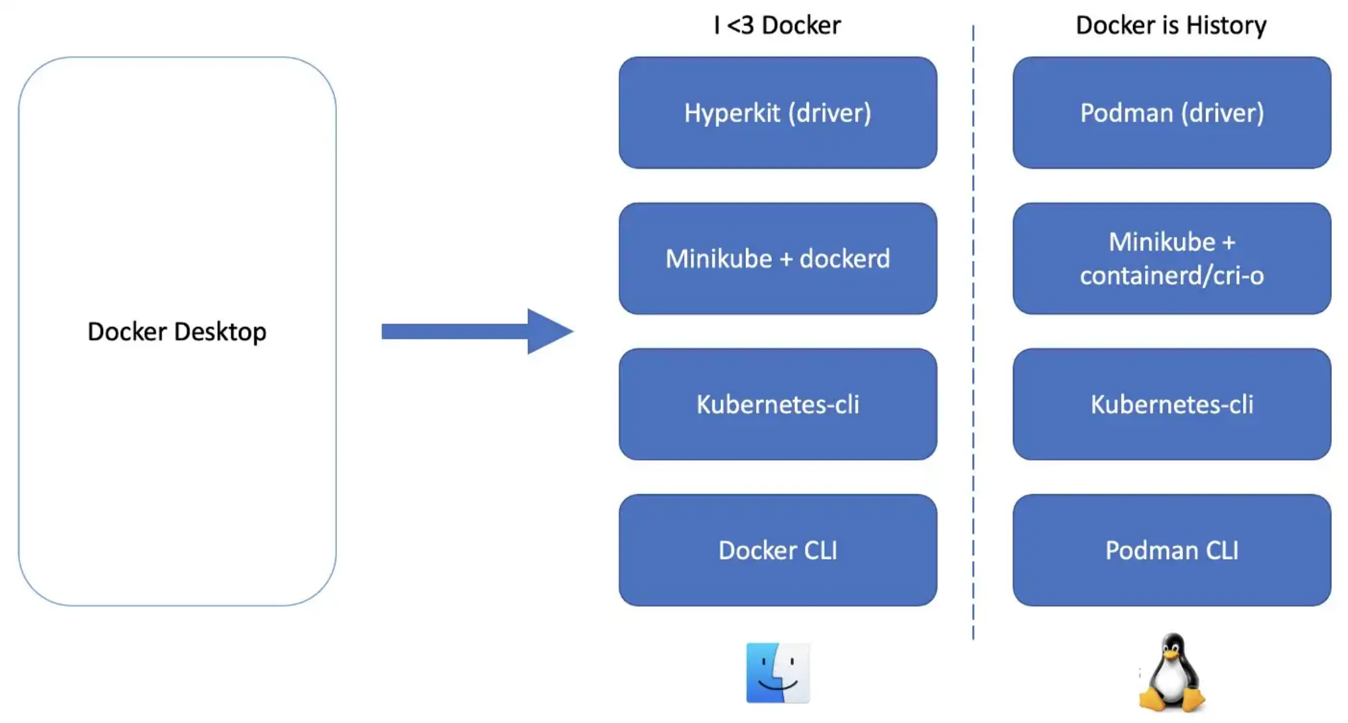
Published in ITNEXT

Abhinav Sonkar Follow
Sep 6, 2021 · 10 min read · Listen

Save

Goodbye Docker Desktop, Hello Minikube!



I have been using Docker Desktop to enable Docker and Kubernetes in Mac for quite some time now. Even though it eats CPU and memory like crazy and makes the fans go wild. With the “in-your-face” popup to force upgrade Docker and the software license change, it was time to look elsewhere for local Kubernetes development needs.

This post is focussed on Mac only. If you have tried this on Linux, let me know how it went.

> Are you on Apple M1 Silicon (ARM64)? Jump to the end of the post for M1 specific instructions.

Uninstall Docker Desktop

Let's start with removing Docker Desktop first.

```
brew uninstall docker
```

This will get rid of not just Docker but also Hyperkit, Docker daemon which allows building images, Docker CLI to interact with the daemon, Kubernetes clusters and kubectl binary (unless you have it deployed separately). If you didn't use Homebrew, then uninstall the tool accordingly.

Let's get these tools back one-by-one.

Install Hyperkit

Hyperkit is still a viable choice for local Kubernetes clusters on Mac. Let's install it.

```
brew install hyperkit
```

Make sure it is installed correctly.

```
❯ hyperkit -v
hyperkit: 0.20200908

Homepage: https://github.com/docker/hyperkit
License: BSD
```

Install Docker CLI

We want to get rid of Docker Desktop but not Docker itself. Docker is still a helpful, open source Container Management tool and if you have a bunch of Dockerfiles to deal with, Docker CLI can be useful.

```
brew install docker
```

> *Note: Do not run `brew install --cask docker`. This will install Docker Desktop and we will be back to where we started!*

This will install the Docker CLI but not the Docker daemon (`dockerd`). You can see this by running `docker info`.

```
❯ docker info
Client:
 Context:    default
 Debug Mode: false

Server:
ERROR: Cannot connect to the Docker daemon at
unix:///var/run/docker.sock. Is the docker daemon running?
```

Install Kubectl

```
brew install kubectl
```

Nothing much to say here.

Install Minikube (and Docker daemon)

With the Hyperkit deployed, we are ready to deploy the Kubernetes cluster and in the process get a Docker daemon as well.

```
brew install minikube
```

Before we get cracking with a Kubernetes cluster, here are some useful things to know:

What driver to use?

In other words, do we deploy Kubernetes in VM, Containers or directly bare-metal? There are various options that can be found here depending on the OS. We will use the Hyperkit driver for Mac (AMD64) and Podman driver for M1 (See end of this post).

What container runtime to use?

Available options — docker, containerd and cri-o. Containerd is a nice choice given Kubernetes itself is moving away from Docker in favor of Containerd. But since we want the Docker daemon to be able to build docker images, let's use Docker.

Set CPU, Memory limits

Just like with Docker Desktop, it is wise to set the correct CPU and memory limits especially if you intend to run many pods.

```
minikube config set cpus 6
minikube config set memory 12g
```

Finally, let's start the Kubernetes cluster.

```
❯ minikube start --kubernetes-version=v1.19.14 --driver=hyperkit --
container-runtime=docker
```

Use the flag `--kubernetes-version` to deploy a specific Kubernetes version. Drop the flag to simply deploy the latest version. I am deploying an older version for my needs.

Here's the output of the above command:

```
😄  minikube v1.23.0 on Darwin 11.5.2
    ▪ MINIKUBE_ACTIVE_DOCKERD=minikube
✨  Using the hyperkit driver based on user configuration
👍  Starting control plane node minikube in cluster minikube
💾  Downloading Kubernetes v1.19.14 preload ...
    > preloaded-images-k8s-v12-v1...: 470.78 MiB / 470.78 MiB
100.00% 6.17 MiB
🔥  Creating hyperkit VM (CPUs=6, Memory=12288MB, Disk=20000MB) ...
❗  This VM is having trouble accessing https://k8s.gcr.io
💡  To pull new external images, you may need to configure a proxy:
https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.19.14 on Docker 20.10.8 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass

❗  /usr/local/bin/kubectl is version 1.22.1, which may have
incompatibilites with Kubernetes 1.19.14.
    ▪ Want kubectl v1.19.14? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and
"default" namespace by default
```

> *If you have `dnsmasq` running locally, there might be failures in DNS resolution within the cluster. You can either uninstall it or add `listen-address=192.168.64.1` to `dnsmasq.conf` . More information can be found here.*

The context is already set. We can check the cluster with `kubectl` as below:

```
❯ minikube kubectl get nodes
NAME       STATUS   ROLES    AGE    VERSION
minikube   Ready    master   7m6s   v1.19.14
```

Since we already deployed kubectl binary, we can use it directly.

At this point, we have a Kubernetes cluster and as we used the Docker driver, the Docker daemon is also running. Before we can use the daemon, let's set the environment variables.

```
eval $(minikube docker-env)
```

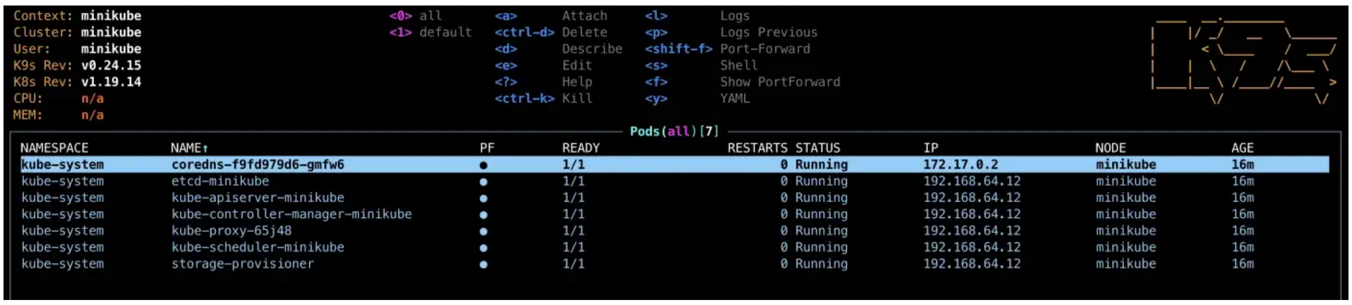
Let's confirm the docker daemon is accessible.

```
❯ docker info
Client:
 Context:    default
 Debug Mode: false

Server:
 Containers: 14
  Running: 14
  Paused: 0
  Stopped: 0
 Images: 10
```

```
  Server Version: 20.10.8
  Storage Driver: overlay2
   Backing Filesystem: extfs
  ...
```

Here is how the Minikube cluster looks with K9S.



```
Context: minikube                  <0> all      <a>       Attach    <l>         Logs
Cluster: minikube                  <1> default  <ctrl-d>  Delete    <p>         Logs Previous
User:    minikube                               <d>       Describe  <shift-f>   Port-Forward
K9s Rev: v0.24.15                               <e>       Edit      <s>         Shell
K8s Rev: v1.19.14                               <?>       Help      <f>         Show PortForward
CPU:     n/a                                    <ctrl-k>  Kill      <y>         YAML
MEM:     n/a
                                          Pods(all)[7]
NAMESPACE    NAME↑                              PF   READY   RESTARTS STATUS    IP             NODE       AGE
kube-system  coredns-f9fd979d6-gmfw6            ●    1/1            0 Running   172.17.0.2     minikube   16m
kube-system  etcd-minikube                      ●    1/1            0 Running   192.168.64.12  minikube   16m
kube-system  kube-apiserver-minikube            ●    1/1            0 Running   192.168.64.12  minikube   16m
kube-system  kube-controller-manager-minikube   ●    1/1            0 Running   192.168.64.12  minikube   16m
kube-system  kube-proxy-65j48                   ●    1/1            0 Running   192.168.64.12  minikube   16m
kube-system  kube-scheduler-minikube            ●    1/1            0 Running   192.168.64.12  minikube   16m
kube-system  storage-provisioner                ●    1/1            0 Running   192.168.64.12  minikube   16m
```

K9S snapshot of freshly installed Minikube cluster

Need Docker Compose?

Install Docker Compose with below command:

```
brew install docker-compose
```

Exposing Services outside Minikube

For local development, it is common to access the services from the laptop via browser or CLI. Port forwarding is always an option but sometimes an Ingress or a Load Balancer is useful. Let’s see how they work with Minikube.

Dealing with Ingress resources

We have a Kubernetes cluster where we can deploy our applications. But how do we access an Ingress resource? Minikube has an answer with addons.

```
❯ minikube addons enable ingress
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v1.0.0-beta.3
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.0
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.0
🔎  Verifying ingress addon...
```

This will deploy the Nginx Ingress Controller. More importantly, it will deploy the Nginx service as `NodePort` and point the Minikube IP to the Ingress directly. Let's find the IP first.

```
❯ minikube ip
192.168.64.12
```

Let's call the above IP on port 80 and we should get the response from Nginx.

```
❯ curl http://192.168.64.12
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Remember that Ingress works on DNS and it should resolve to the Minikube IP. If one of the backend services calls that DNS, it will fail unless explicitly configured. Another addon to the rescue!

```
❯ minikube addons enable ingress-dns
    ▪ Using image cryptexlabs/minikube-ingress-dns:0.3.0
```

This starts a DNS server on the Minikube IP within the Kubernetes cluster. A custom resolver is also required to force the DNS resolution of a custom TLD (like .test, do not use .local) to be redirected to the DNS server started above. The Minikube Ingress DNS documentation explains this very well. Follow the steps there and you will have a working Ingress setup with a custom DNS.

Load Balancer Services

Wouldn't it be great to deploy services of type Load Balancer and access it from the host machine just like you would when deploying in Cloud? Thanks to the `metallb` addon this is quite straightforward.

```
minikube addons enable metallb
```

This will deploy two more pods which are responsible for assigning an external IP to the Load Balancer services. Without this, the services will have `pending` external IP forever.

There is one more step before `metallb` can be used. By default, `metallb` has no way to know which range of IPs are available for assigning to Load Balancer services. Run below command to provide a range.

```
❯ minikube addons configure metallb
-- Enter Load Balancer Start IP: 192.168.64.5
-- Enter Load Balancer End IP: 192.168.64.15
    ▪ Using image metallb/speaker:v0.9.6
    ▪ Using image metallb/controller:v0.9.6
✅  metallb was successfully configured
```

Based on your Minikube IP, assign a small range of IPs that include the Minikube IP. Now whenever you deploy a Load Balancer service, an IP from this range will be assigned.

Other Issues

Logging In to Remote Registry

You may still have the old `~/.docker/config.json` with `credsStore` set to either `osxkeychain` or `desktop`. This will not work in the new setup. To fix this, let's install Docker Credential Helper.

```
brew install docker-credential-helper
```

The credentials will be stored in MacOS Keychain as before. If this does not work a quick fix is to remove the `~/.docker/config.json` file and login to the registries again.

Retaining Docker Images and Persistent Volume Claims

Added 7th Sept 2021

A nice thing with Docker Desktop was that you could shutdown the Kubernetes cluster, start it again at a later date and run your pods with the same Docker images and persistent volumes. This is useful for example when running a local database in Kubernetes. Having the persistent volume available across restarts is handy.

With Minikube, if I shutdown the cluster (and the Hyperkit VM) with `minikube stop`, it removes the Docker images and all persistent volumes. That is annoying. Fortunately, Minikube provides a way to prevent the deletion. Instead of stopping the Kubernetes cluster and Hyperkit VM, we can *pause* it.

```
minikube pause
```

This command terminates the Kubernetes cluster but does not remove the Hyperkit VM. This frees up even more CPU while still retaining all Docker images and persistent volumes. But wait it gets even better! It does NOT stop the docker daemon. So you can continue using the Docker CLI, just don't forget to set the docker environment with `eval $(minikube docker-env)`.

When you want to resume working on the Kubernetes cluster, run below command:

```
minikube unpause
```

And you will have all the system pods back including the addons. This works even across laptop restarts!

Bind Mounting in Docker Containers

Added 7th Sept 2021

The good folks over at Reddit pointed out that bind mounts `(-v)` in Docker containers does not work out of the box with Minikube and Docker setup. This is a common operation with Docker containers and should just work.

Because of the intermediate Hyperkit VM, mounting a volume is a two-step operation. First, let's mount the disk on the laptop to the Hyperkit VM.

```
minikube mount /myvolume:/test
```

This will mount the local folder `/myvolume` at the path `/test` within the Hyperkit VM. The process remains active so you should leave this terminal alone.

In another terminal, run the Docker container and bind mount the `/test` volume to a path within the container.

```
docker run --rm -it -v /test:/inside busybox /bin/sh
```

This will mount the `/test` volume on the Hyperkit VM inside the container at the path `/inside` . Effectively this makes all files and folders under `/myvolume` on the laptop inside the container in read-write mode. Sweet!

What about Apple Silicon M1?

Added 5th Feb 2022

Last month I finally got my hands on the Apple M1! Since then I have been trying to figure out a way to run Minikube without using Docker desktop… until now :)

Apple M1 based on ARM64 has some challenges. Right out of the bat, Hyperkit does not work on M1 and probably never will. We need an alternative tool to create the virtual machine. Enter Podman.

As per their website, Podman is straight replacement of Docker. Literally `alias docker=podman` . The exciting news is Podman has added support for Apple M1! We will use the Podman machine (not unlike docker machine) feature to make ourselves a virtual machine. Under the hood, both Podman and Docker Desktop use QEMU to do their magic.

First install Podman.

```
brew install podman
```

Create a virtual machine with desired specs.

```
❯ podman machine init --cpus 6 --memory 12288 --disk-size 50 --
image-path next
Downloading VM image: fedora-coreos-35.20220131.1.0-
qemu.aarch64.qcow2.xz: done
Extracting compressed file

❯ podman machine start
INFO[0000] waiting for clients...
INFO[0000] listening tcp://127.0.0.1:7777
INFO[0000] new connection from  to
/var/folders/2x/wz0vrff903q0p3lpn959b8zw0000gn/T/podman/qemu_podman-
machine-default.sock
Waiting for VM ...
Machine "podman-machine-default" started successfully
```

A small tweak is required to use the root destination of Podman service to make it work with Minikube. Without this tweak, Minikube will fail with "Error: the requested cgroup controller `cpu` is not available: OCI runtime error".

```
❯ podman system connection default podman-machine-default-root

❯ podman system connection list
Name                          Identity
URI
podman-machine-default        /Users/x/.ssh/podman-machine-default
ssh://core@localhost:57623/run/user/1000/podman/podman.sock
podman-machine-default-root*  /Users/x/.ssh/podman-machine-default
ssh://root@localhost:57623/run/podman/podman.sock
```

With that Podman is set. Next step is to start the Minikube cluster using Podman driver. Just run the below command.

```
❯ minikube start --kubernetes-version=v1.22.5 --driver=podman --
container-runtime=cri-o -p minipod
😄  minikube v1.28.0 on Darwin 12.6.1 (arm64)
🆕  Kubernetes 1.25.3 is now available. If you would like to
upgrade, specify: --kubernetes-version=v1.25.3
✨  Using the podman (experimental) driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
E1218 00:06:49.506418   69295 cache.go:203] Error downloading kic
artifacts:  not yet implemented, see issue #8426
🔄  Restarting existing podman container for "minikube" ...
🎁  Preparing Kubernetes v1.22.5 on CRI-O 1.24.3 ...
E1218 00:06:57.412980   69295 start.go:130] Unable to get host IP:
RoutableHostIPFromInside is currently only implemented for linux
```

```
🔗  Configuring CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
```

Open in app ↗

And with that, we have a Minikube cluster running on Apple M1 without Docker desktop!

Conclusion

After spending a few hours on a Sunday afternoon, I am quite happy with the setup. We got rid of Docker Desktop and replaced it with Hyperkit (Podman for M1) and Minikube. We can still use the Docker API to manage Dockerfiles and deploy apps in a local Kubernetes cluster. Most importantly, the laptop is happily chugging along and the extra resources can be used by Slack, Notion and other Electron apps ;-)

Resources

1. Introduction to Container Runtimes
2. Minikube documentation
3. Podman documentation

Docker Desktop   Kubernetes   Docker   Container Runtime   Minikube

2.3K | 22 | •••