You can use Tailscale with Kubernetes, you know

tailscale.com/blog/kubecon-21

Given that this week is the epic all-things-cloud-native reunion in LA, we thought we might crash your little party and mention that Tailscale already works well with containers and Kubernetes. Many of us here at Tailscale used to work on Kubernetes, and keep it close to our hearts even if we're not at KubeCon this week (and sorry, we love YAML, but use HuJSON now).

Making Tailscale work well with container workloads is something we've been working on for a long time. We're far from done, but a lot of things have changed since the last in-person KubeCon (for one, most of Tailscale).

Tailscale has a managed Docker image

Tailscale now has a published Docker image, that Tailscale manages and builds from source. It's available in Docker Hub and GitHub Packages, just run `docker pull tailscale/tailscale:latest` or `docker pull ghcr.io/tailscale/tailscale:latest`.

This is not yet available in other public container registries. Please let us know what else you might need from this image.

Tailscale can run without any permissions inside a container

First thing's first, getting Tailscale working in containers! Normally, Tailscale on Linux uses the `/dev/net/tun` driver as a network interface, but that's not always available in containers, especially for hosted platforms. You use TUN when Tailscale routes packets directly from itself to the kernel, making `tailscale0` appear alongside `eth0` or `wifi0` for Ethernet or Wi-Fi interfaces. Your application sends packets using regular sockets, which the kernel sends up to `tailscale0` for processing.

As of v1.6, Tailscale can run without the need for a TUN device using userspace networking. This mode drops the requirement for Linux capabilities `CAP_NET_RAW` and `CAP_NET_ADMIN`. `CAP_NET_RAW` is used for accessing privileged socket operations and `CAP_NET_ADMIN` is used for configuring a network; by removing these, if an attacker

were to discover and exploit a vulnerability in Tailscale, they couldn't use these capabilities in your system. With our latest release, Tailscale v1.16, Tailscale also ships with a HTTP proxy server in addition to the SOCKS proxy support added in v1.6. This lets you run Tailscale on container and serverless platforms like Heroku, Google Cloud Run, and GitHub Actions.

We know we still have some work to do here. Userspace networking doesn't allow you to use an exit node or connect to a subnet router, but does work if the container itself is an exit node or subnet router. *Update: You can use an exit node or connect to a subnet router in userspace networking mode in Tailscale v1.20 or later.*

Tailscale lets you add containers to your network as ephemeral nodes

For workloads that appear and disappear, like containers and functions, you want an authentication key that can be reused across workloads, but that won't make you deal with that long list of workloads as if they were pets instead of cattle. So, connect an ephemeral node to your tailnet using an ephemeral auth key. You can't yet add an ACL tag to an authentication key. *Update: Auth keys can now include an ACL tag binding.*

Ephemeral keys let you authenticate a device to your network and when the workload is torn down, the device is removed from your network. This is useful if you want to, say, connect your build container to your internal package registry over Tailscale.

Tailscale can run in a sidecar container alongside other workloads

If for some reason, you can't run Tailscale in the same container as your workload (due to security reasons, or because it's a vendor provided image), then you can still use Tailscale! Just run it as a sidecar container for your pod in Kubernetes. This does need `CAP_NET_ADMIN` if you need outbound connections to reach back to the tailnet, but not for inbound connections from other machines on the network. Check out our repo for a sample sidecar container.

In Tailscale v1.16, we also made it possible for you to store the persistent state of Tailscale, the node key, and the machine key as Kubernetes secrets.

Tailscale can act as a proxy

If you're running a workload on a Kubernetes cluster that needs to be shared with others in your network, you can use Tailscale to make that workload accessible and still use MagicDNS to access your workloads. Check out our repo for a sample proxy container.

This means that you can easily share and connect to a workload running in a Kubernetes cluster with other parts of your network, for example to have your production workload contact your product database over Tailscale or expose an internal-only service over Tailscale and not the public internet.

For more information on using Tailscale with Kubernetes, see Tailscale on Kubernetes.