Introducing the Tailscale Universal Docker Mod

Xe Iaso (they/them) Tailscalar on April 14, 2023

Imagine a world where you could add applications to your tailnet the same way you add machines to it. This would mean that `http://wiki` would go to your internal wiki, `http://code` would take you to an IDE, and `http://chat` would take you to your internal chat server. This is the world that Tailscale lets you create, but historically the details on how you would actually do this are left as an exercise for the reader.

Today, we're introducing a new way to add Tailscale to your Docker containers: our brand new universal Docker mod. This lets you add Tailscale to any Docker container based on linuxserver.io images. This lets you have applications join your tailnet just as easily as machines can. You can set up a wiki on `http://wiki`, an IDE at `http://code`, and a chat server at `http://chat` and have them all be accessible over your tailnet. You can even use this to expose your internal applications to the public internet with Funnel.

<**Xe**> You can even use this to SSH into containers!

<**Aoi**> You can *what* into a container?

<**Xe**> Yep! Tailscale SSH lets you SSH into containers when you enable the `TAILSCALE_USE_SSH` setting and permit access in the ACLs. This is

a great way to get into a container without having to SSH into the docker host and run `docker exec -it <container> bash`.

To add this to your existing Docker containers with linuxserver.io images, add the following environment variables to your docker-compose.yml file:

```
- DOCKER_MODS=ghcr.io/tailscale-dev/docker-mod:main
# tailscale configuration

# make sure this is persisted in a volume
- TAILSCALE_STATE_DIR=/var/lib/tailscale
- TAILSCALE_SERVE_MODE=https
- TAILSCALE_SERVE_PORT=80
- TAILSCALE_USE_SSH=1
- TAILSCALE_HOSTNAME=wiki

## uncomment to enable funnel
## remember that if you do, it's exposed to the internet, so be ca
#- TAILSCALE_FUNNEL=on

# replace this with your authkey from the admin panel
- TAILSCALE_AUTHKEY=tskey-auth-hunter2CNTRL-hunter2hunter2
```

This will add Tailscale to your container so that you can access it over your tailnet. If you run `docker compose up -d` with the authkey changed out for a valid authkey, you'll be able to access your apps over Tailscale.

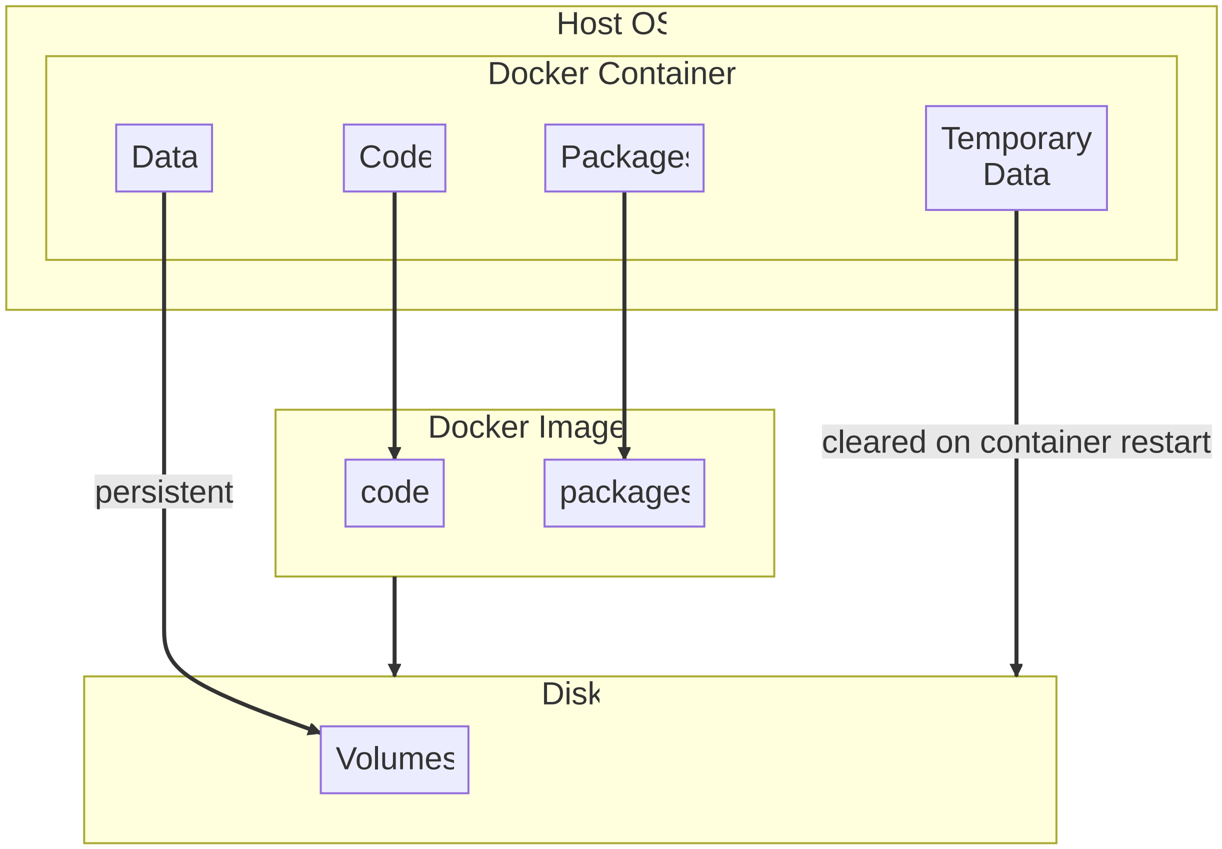
Image generated by Ligne Claire v1.5, prompt: flat color, shipyard, containers, mountains, no humans, space needle

Docker and Docker mods

Docker allows you to create snapshots of operating system installs with a given state, such as "having the Go compiler available" or "install this program and all its dependencies" and distribute those preconfigured images on the Internet. When you consume the same Docker image at two time intervals T0 and T1, you get the same image with the same code, just as you expect.

When a Docker container is run, it usually runs on top of an ephemeral filesystem that gets destroyed when the container is stopped. This means that restarting the container will reset it back to the state that was there when the image was created. This is normally convenient when working on applications that make temporary changes to the filesystem, such as an image converter that uses temporary files to do the conversion logic.

This is less convenient when you want to run things like database servers in Docker. However, most of the time when you do things that need persistent state, that persistent state is usually limited to a single file or directory. Docker provides external persistent state with volumes. They're basically directories that are plunked into the container at runtime, but it maintains the state between container runs. This is great for things like databases because you wouldn't want to lose all your data when you restart the container.



So, from here we can create a hierarchy for docker and statefulness. You expect docker containers to have state for *data*, and you also expect the docker container to be running the same code every time you run the same image. You don't expect anything else to be running, everything is deterministic at T0, T1, or TN.

<**Xe**> This is a valid hierarchy because it's what you expect from docker. You expect the same code to run every time you run the same image.

What is humor?

Humor is a complicated concept that is almost universal throughout human cultures. It's a way of conveying concepts like absurdity, irony, the absurdity of irony, and normally frustrating things in ways that aren't quite as much of a downer. It's really about being able to communicate subtle things like common errors that everyone makes when learning things (such as English and its rule of all the rules having exceptions, even for the exceptions). It's also a tool that you can use to help describe the abstract and nonphysical things like emotions, feelings, ideas, the human condition, and how Kubernetes works.

<**Xe**> Humor is also really hard to convey properly in a written medium. This is even more difficult when the humor is about technology, which is usually hard to understand in the first place. I'm going to try to explain the humor in this article with these asides so that y'all can follow along, but if you already get why this is funny it may ruin the joke for you. Sorry!

In his famous presentation Reverse emulating the NES, fellow philosopher in arms tom7 introduced the idea of a type of humor called "invalid hierarchies". In this he does rather abusrd things to an NES using a custom circuit board and a raspbery pi to allow him to (among other things) run an SNES emulator on the NES. This video is quite possibly one of my favorite technical communication videos and is a huge influence to how I write humorous things for this blog.

<**Xe**> This creates an invalid hierarchy because you expect the NES to only run 8-bit NES games, but not 16-bit SNES games. This is funny. If you've never seen that video before, it's well worth a watch.

Another example of an invalid hierarchy is my April Fool's Day post Using Tailscale without using Tailscale. You'd expect to have to use Tailscale if you want to use Tailscale, but "using Tailscale without using Tailscale" creates an invalid hierarchy in the mind of the reader. This is also funny.

Docker mods

Docker mods let you install extra packages and services into containers at runtime. If the `ONBUILD` hook lets you run a series of commands when an image is built, you can think of docker mods as a missing `ONRUN` hook that lets you customize an image at runtime.

<**Xe**> This creates an invalid hierarchy because we think about the code in a container being deterministic between invocations and this allows you to make something *nondeterministic*. This is funny.

Docker mods and s6

At a high level, a docker mod is a series of files that add additional instructions to the start phase of a docker container. It works because the linuxserver.io containers preinstall s6 via s6-overlay and then start it in the background to manage the lifecycle of services in the container.

<**Xe**> This is also funny because usually Docker containers aren't supposed to have multiple processes running in them for simplicity, but it turns out that when you want to do things like put your wiki seamlessly on

your tailnet, you want to have multiple processes running. This is another invalid hierarchy because you expect the container to only have one process running, but it has multiple with a service manager, just like the host OS.

When I made the docker mod, I had to create a few s6 services to help it run:

- One to set a list of packages that Tailscale needs to run (jq to process some data from the packages server, and iptables to configure the firewall inside the container for Tailscale to run in a TUN device).
- One to download Tailscale to the container.
- One to start the Tailscale node agent `tailscaled`.
- One to authenticate you to the tailnet with `tailscale up` and set other settings like `tailscale serve`.

<**Xe**> This is also hilarious because this roughly mirrors the process that you have to do on your host OS to get Tailscale running. This is another layer of invalid hierarchy because you expect containers to ship with all the software they need, but here is this container that needs to download software at runtime. This is funny because it's like a container that needs to download software at runtime, just like your host OS. As above, so below, eh?

Each of these is connected together like this (arrows indicate dependencies):

<**Xe**> If you've ever worked deeply with the Heroku ecosystem, you can think about Docker mods as akin to all of the hilarous hacks you can do with buildpacks at dyno boot time.

Configuration

The Docker mod exposes a bunch of environment variables that you can use to configure it. You can see the full list of environment variables in the documentation, but here are the important ones:

Environment Variable	Description	Example
`DOCKER_MODS`	The list of additional mods to layer on top of the running container, separated by pipes.	`ghcr.io/tailscale-dev/docker-mod:main`

Environment Variable	Description	Example
`TAILSCALE_STATE_DIR`	The directory where the Tailscale state will be stored, this should be pointed to a Docker volume. If it is not, then the node will set itself as ephemeral, making the node disappear from your tailnet when the container exits.	`/var/lib/tailscale`
`TAILSCALE_AUTHKEY`	The authkey for your tailnet. You can create one in the admin panel. See here for more information about authkeys and what you can do with them.	`tskey-auth-hunter2CNTRL-hunter2hunter2`
`TAILSCALE_HOSTNAME`	The hostname that you want to set for the container. If you don't set this, the hostname of the node on your tailnet will be a bunch of random hexadecimal numbers, which many humans find hard to remember.	`wiki`
`TAILSCALE_USE_SSH`	Set this to `1` to enable SSH access to the container.	`1`
`TAILSCALE_SERVE_PORT`	The port number that you want to expose on your tailnet. This will be the port of your DokuWiki, Transmission, or other container.	`80`
`TAILSCALE_SERVE_MODE`	The mode you want to run Tailscale serving in. This should be `https` in most cases, but there may be times when you need to enable `tls-terminated-tcp` to deal with some weird edge cases like HTTP long-	`https`

Environment Variable	Description	Example
	poll connections. See here for more information.	
`TAILSCALE_FUNNEL`	Set this to `true`, `1`, or `t` to enable funnel. For more information about the accepted syntax, please read the strconv.ParseBool documentation in the Go standard library.	`on`

Something important to keep in mind is that you really should set up a separate volume for Tailscale state. Here is how to do that with the docker commandline:

```
docker volume create dokuwiki-tailscale
```

Then you can mount it into a container by using the volume name instead of a host path:

```
docker run \
  ... \
  -v dokuwiki-tailscale:/var/lib/tailscale \
  ...
```

If you want to use kernel networking mode, you will need to add the `NET_ADMIN` and `NET_RAW` capabilities to the container, as well as pass the `/dev/net/tun` device into the container. Here is an example of how to do that with the docker commandline:

```
docker run \
  ... \
  --cap-add=NET_ADMIN \
  --cap-add=NET_RAW \
  --device=/dev/net/tun \
  ...
```

In a `compose.yaml` file, it will look like this:

```yaml
version: '2.1'
services:
  dokuwiki:
    image: lscr.io/linuxserver/dokuwiki:latest
    volumes:
      - /dev/net/tun:/dev/net/tun
    cap_add:
      - NET_ADMIN
      - NET_RAW
    # ...
```

This can be useful when you are running applications on your tailnet *without* tailscale serve, and you want the underlying service to know the exact remote IP address (such as when running a Minecraft server).

Fun things you can do

Normally when I write these articles, I tend to give you one functional example so that you can fill in the blanks here. This time, I want to give you a few functional and genuinely useful examples so that you can get started with our Docker mod right away.

If you want to test this with a simple command-line shell, you can run this docker command to create a volume for Tailscale state, and then run a container with the Docker mod installed:

```
docker volume create trap-sun-state
```

<**Xe**> `trap-sun` is the name of the container that we will be running. You can name it whatever you want, but you should use the same name in both your volume and your container. I'm setting the name here in case you get stuck and need to arbitrarily kill the container with `docker kill trap-sun`.

```
docker run \
  --rm \
  -v trap-sun-state:/var/lib/tailscale \
  -e TAILSCALE_STATE_DIR=/var/lib/tailscale \
  -e TAILSCALE_SERVE_PORT=3000 \
  -e TAILSCALE_SERVE_PORT=3000 \
  -e TAILSCALE_SERVE_MODE=https \
  -e TAILSCALE_FUNNEL=on \
  -e TAILSCALE_USE_SSH=1 \
  -e TAILSCALE_HOSTNAME=trap-sun \
  -e TAILSCALE_AUTHKEY=tskey-auth-hunter2CNTRL-hunter2hunter2 \
  -e DOCKER_MODS=ghcr.io/tailscale-dev/docker-mod:main \
  --name trap-sun \
  -it \
  --cap-add=NET_ADMIN \
  --cap-add=NET_RAW \
  -v /dev/net/tun:/dev/net/tun \
  lsiobase/alpine:3.17 \
  sh
```

<**Xe**> You can also base your Docker images on the `lscr.io/linuxserver/baseimage-alpine:3.17` image, which is a minimal Alpine Linux with Docker mod support. This can be used to adapt your existing containers into nodes on your tailnet. You can also use Ubuntu with `lscr.io/linuxserver/baseimage-ubuntu:jammy` as the base image. The cloud's the limit!

DokuWiki

If you want to set up a wiki for your tailnet with DokuWiki, you can use this Docker compose file:

```yaml
# docker-compose.yaml
version: '2.1'
services:
  dokuwiki:
    image: lscr.io/linuxserver/dokuwiki:latest
    container_name: dokuwiki
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Etc/UTC
      - DOCKER_MODS=ghcr.io/tailscale-dev/docker-mod:main
      # tailscale information
      - TAILSCALE_STATE_DIR=/var/lib/tailscale
      - TAILSCALE_SERVE_PORT=80
      - TAILSCALE_SERVE_MODE=https
      ## uncomment to enable funnel, may be a bad idea for some us
      #- TAILSCALE_FUNNEL=on
      - TAILSCALE_USE_SSH=1
      - TAILSCALE_HOSTNAME=wiki
      - TAILSCALE_AUTHKEY=tskey-auth-hunter2CNTRL-hunter2hunter2

    volumes:
      - dokuwiki-data:/config
      - dokuwiki-tailscale:/var/lib/tailscale
    restart: unless-stopped

volumes:
  dokuwiki-data:
  dokuwiki-tailscale:
```

Then use `docker compose up -d` to start the DokuWiki container with Tailscale grafted in. You can then access your DokuWiki instance at `https://wiki.yourtailnet.ts.net`. You will want to do the setup wizard, and then you can start using your own private wiki!

Your private cloud development environment with code-server

Want to have all the fun of GitHub Codespaces without having to use GitHub's servers for development? Set up your own private cloud with code-server and Tailscale!

```
version: '2.1'
services:
  code-server:
    image: lscr.io/linuxserver/code-server:latest
    container_name: code-server
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Etc/UTC
      - PASSWORD=hunter2
      - PROXY_DOMAIN=code.shark-harmonic.ts.net
      - DOCKER_MODS=ghcr.io/tailscale-dev/docker-mod:main|ghcr.io/
      # tailscale information
      - TAILSCALE_STATE_DIR=/var/lib/tailscale
      - TAILSCALE_SERVE_PORT=8443
      - TAILSCALE_SERVE_MODE=tls-terminated-tcp
      - TAILSCALE_USE_SSH=1
      - TAILSCALE_HOSTNAME=code
      - TAILSCALE_AUTHKEY=tskey-auth-hunter2CNTRL-hunter2hunter2
    volumes:
      - code-server-data:/config
      - code-server-tailscale:/var/lib/tailscale
    restart: unless-stopped

volumes:
  code-server-data:
  code-server-tailscale:
```

Then use `docker compose up -d` to start the code-server container with Tailscale grafted in. You can then access your code-server instance at `https://code.shark-harmonic.ts.net`. You may want to change the password from `hunter2` to something more secure.

code-server also has support for cloning repositories from GitHub directly, so with this you can get started hacking on a project on one machine, then seamlessly pick up where you left off on another! You can start hacking at something in your office and then walk over to the local Tim Horton's to finish it up!

There's a bunch of other containers in the linuxserver.io fleet, you can use Tailscale with those as well. You can also check out Awesome-LSIO for more ideas!

At Tailscale, we want to recreate the Internet around the idea of small, trusted networks with your friends, family, and coworkers. When you set up applications on your tailnet like this, you can slowly start to use your own private infrastructure instead of relying on the public Internet. This is a great way to start using Tailscale, and we hope that you will find this Docker mod useful.

If you have any questions, feel free to reach out to us on Twitter or the Fediverse. We are always happy to help!