Using WireGuard to create a network overlay

In this article

You can create an overlay network between your runner and a service in your private network.

Using WireGuard to create a network overlay

If you don't want to maintain separate infrastructure for an API Gateway, you can create an overlay network between your runner and a service in your private network, by running WireGuard in both places.

There are various disadvantages to this approach:

- To reach WireGuard running on your private service, you will need a well-known IP address and port that your workflow can reference: this can either be a public IP address and port, a port mapping on a network gateway, or a service that dynamically updates DNS.
- WireGuard doesn't handle NAT traversal out of the box, so you'll need to identify a way to provide this service.
- This connection is one-to-one, so if you need high availability or high throughput you'll need to build that on top of WireGuard.
- You'll need to generate and securely store keys for both the runner and your private service. WireGuard uses UDP, so your network must support UDP traffic.

There are some advantages too, as you can run WireGuard on an existing server so you don't have to maintain separate infrastructure, and it's well supported on GitHub-hosted runners.

Example: Configuring WireGuard

This example workflow configures WireGuard to connect to a private service.

For this example, the WireGuard instance running in the private network has this configuration:

- Overlay network IP address of `192.168.1.1`
- Public IP address and port of `1.2.3.4:56789`
- Public key `examplepubkey1234...`

The WireGuard instance in the GitHub Actions runner has this configuration:

- Overlay network IP address of `192.168.1.2`
- Private key stores as an GitHub Actions secret under `WIREGUARD_PRIVATE_KEY`

```yaml
name: WireGuard example

on:
  workflow_dispatch:

jobs:
  wireguard_example:
    runs-on: ubuntu-latest
    steps:
      - run: sudo apt install wireguard

      - run: echo "$" > privatekey

      - run: sudo ip link add dev wg0 type wireguard

      - run: sudo ip address add dev wg0 192.168.1.2 peer 192.168.1.1

      - run: sudo wg set wg0 listen-port 48123 private-key privatekey peer
examplepubkey1234... allowed-ips 0.0.0.0/0 endpoint 1.2.3.4:56789

      - run: sudo ip link set up dev wg0

      - run: curl -vvv http://192.168.1.1
```

For more information, see WireGuard's Quick Start, as well as "Using secrets in GitHub Actions" for how to securely store keys.

Using Tailscale to create a network overlay

Tailscale is a commercial product built on top of WireGuard. This option is very similar to WireGuard, except Tailscale is more of a complete product experience instead of an open source component.

Its disadvantages are similar to WireGuard: The connection is one-to-one, so you might need to do additional work for high availability or high throughput. You still need to generate and securely store keys. The protocol is still UDP, so your network must support UDP traffic.

However, there are some advantages over WireGuard: NAT traversal is built-in, so you don't need to expose a port to the public internet. It is by far the quickest of these options to get up and running, since Tailscale provides an GitHub Actions workflow with a single step to connect to the overlay network.

For more information, see the Tailscale GitHub Action, as well as "Using secrets in GitHub Actions" for how to securely store keys.