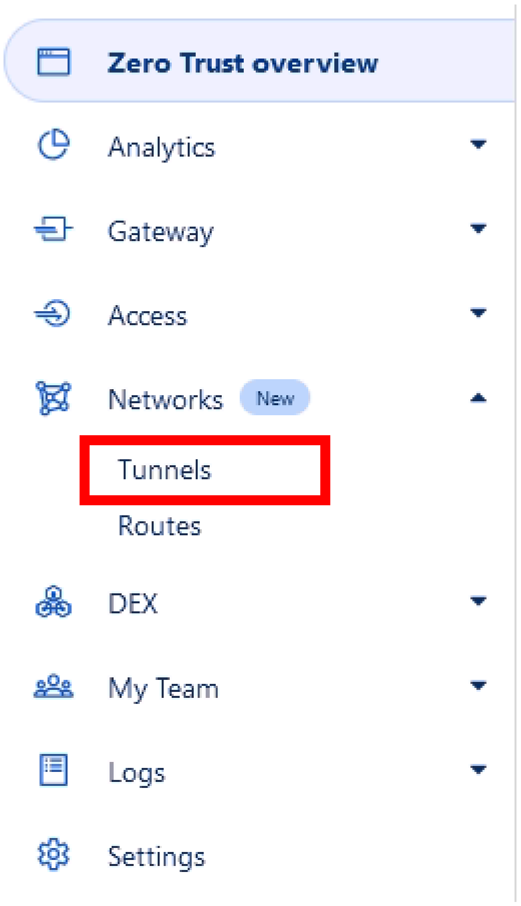
Documentation

Docker Compose Deployments

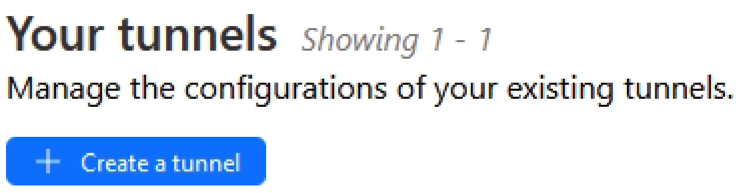
Backup Architecture

Cloudflare Tunnel Configuration

1. Navigate and sign in to the Cloudflare Zero Trust dashboard.
2. Using the sidebar, navigate to `Networks -> Tunnels`.



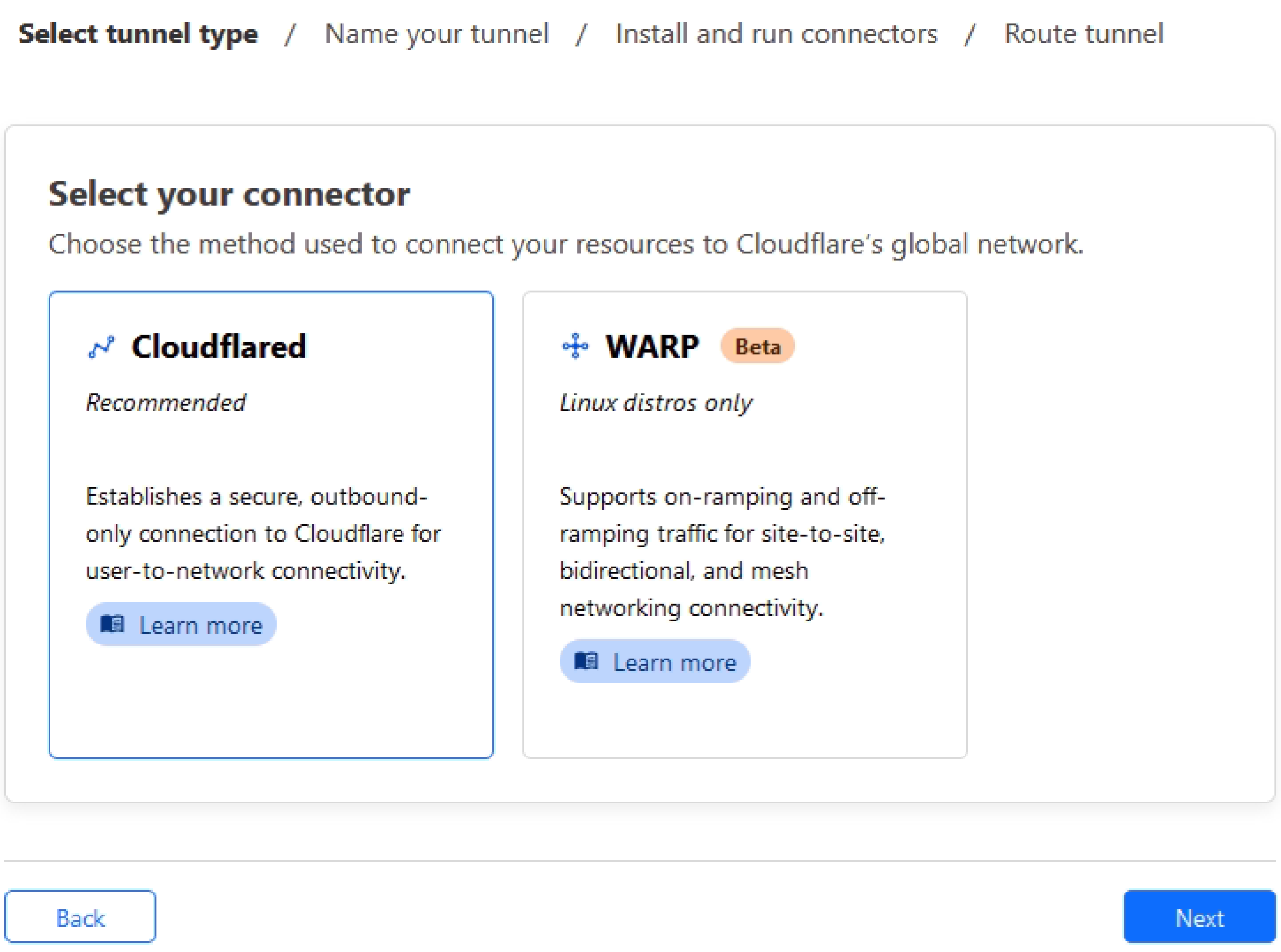
3. Click the blue 'Create a tunnel' button.



Your tunnels *Showing 1 - 1*

Manage the configurations of your existing tunnels.

4. Select the default 'Cloudflared' connector type, and then click 'Next'.


Select tunnel type / Name your tunnel / Install and run connectors / Route tunnel

Select your connector

Choose the method used to connect your resources to Cloudflare's global network.

Cloudflared

Recommended

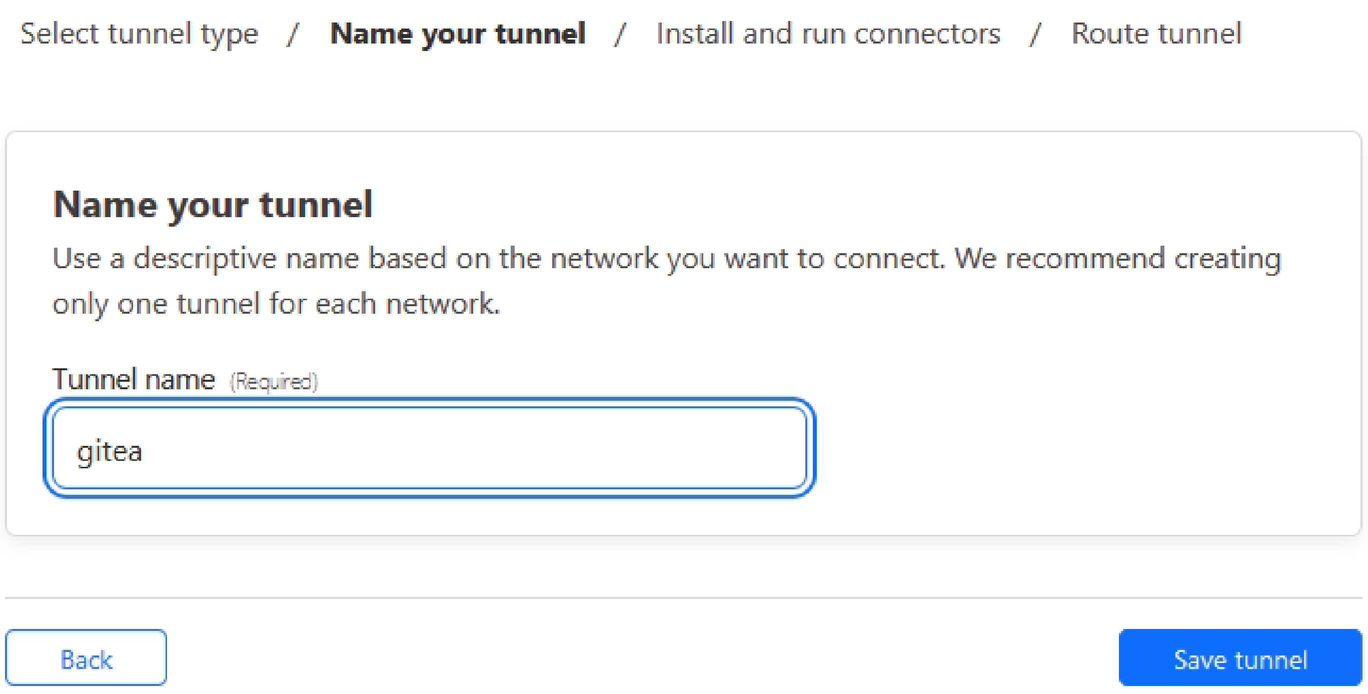
Establishes a secure, outbound-only connection to Cloudflare for user-to-network connectivity.

WARP Beta

Linux distros only

Supports on-ramping and off-ramping traffic for site-to-site, bidirectional, and mesh networking connectivity.

5. Give the tunnel a name, then click 'Save tunnel'.


Select tunnel type / Name your tunnel / Install and run connectors / Route tunnel

Name your tunnel

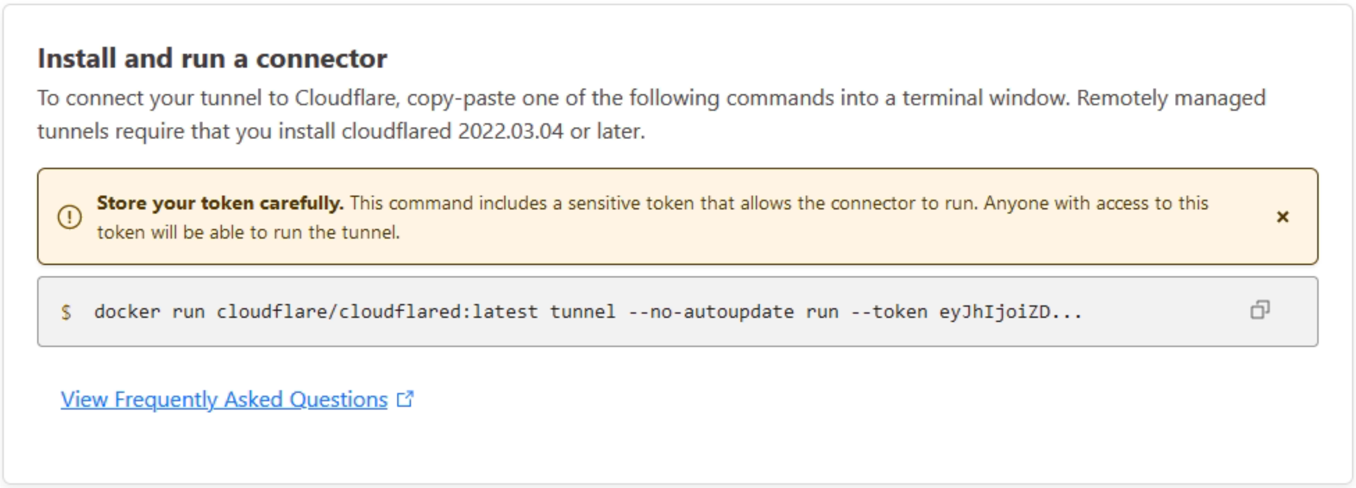
Use a descriptive name based on the network you want to connect. We recommend creating only one tunnel for each network.

Tunnel name (Required)

gitea

You will then be presented with a list of connector installation options.

6. Copy the token and use it with your preferred connector. For Docker instructrions, see Docker Compose Template



Install and run a connector

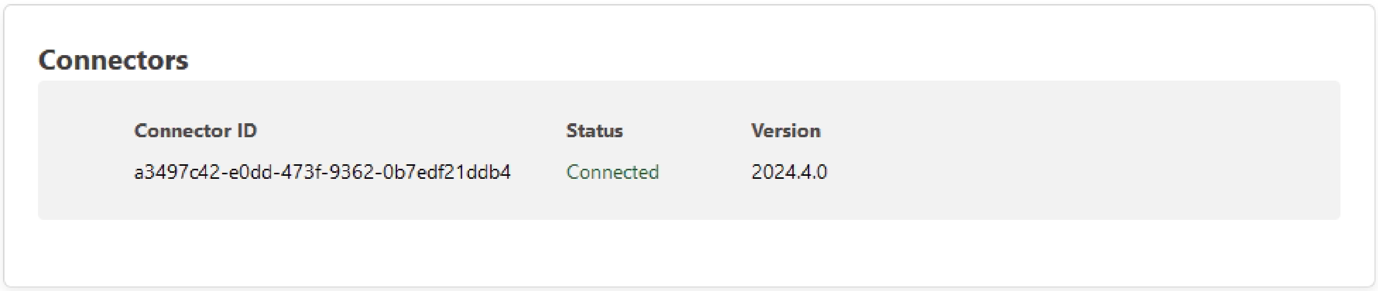
To connect your tunnel to Cloudflare, copy-paste one of the following commands into a terminal window. Remotely managed tunnels require that you install cloudflared 2022.03.04 or later.

Store your token carefully. This command includes a sensitive token that allows the connector to run. Anyone with access to this token will be able to run the tunnel.

```
$ docker run cloudflare/cloudflared:latest tunnel --no-autoupdate run --token eyJhIjoiZD...
```

View Frequently Asked Questions

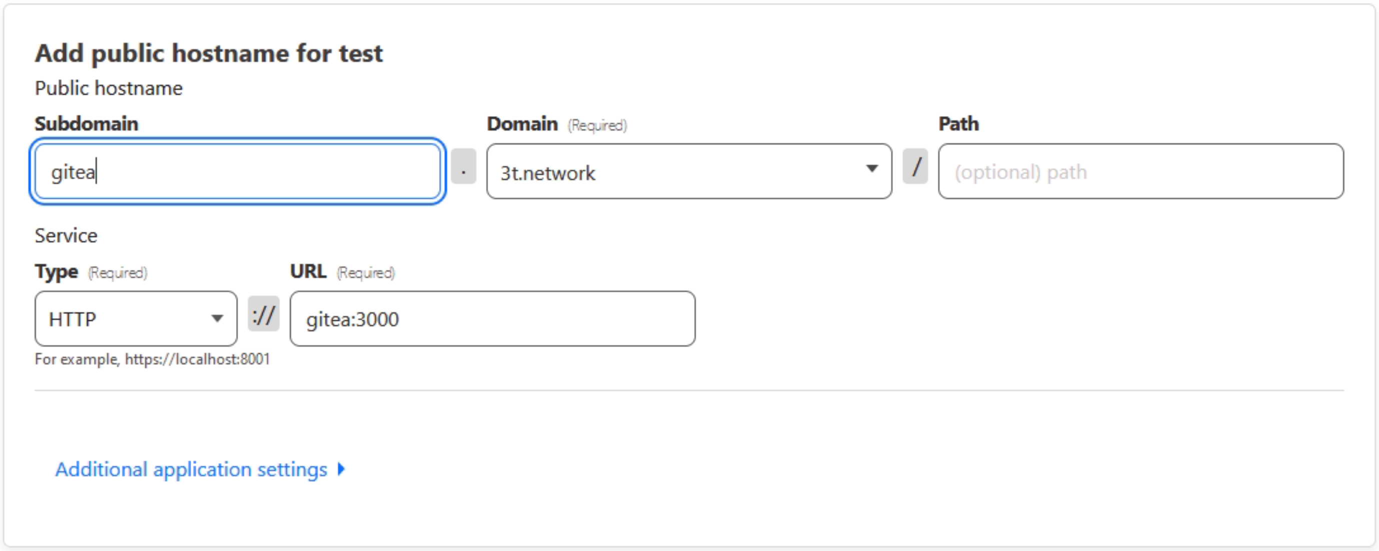
8. Once connected, your connector should appear at the bottom of the page. Click 'Next'.



Connectors

Connector ID	Status	Version
a3497c42-e0dd-473f-9362-0b7edf21ddb4	Connected	2024.4.0

9. Finally, assign your service a subdomain, and point it to the backend.



Add public hostname for test

Public hostname

Subdomain	Domain (Required)	Path
gitea	3t.network	(optional) path

Service

Type (Required)	URL (Required)
HTTP	gitea:3000

For example, https://localhost:8001

Additional application settings

Please note, when using Docker networking (as per Docker Compose Template), there is no requirement to 'expose' the port with a port mapping. You can use the name of the container, as defined in the `docker-compose.yml` file with the appropriate listening port.

Docker Compose Template

Guidelines

These guidelines are suggested in order to maximise reliability of hosted services.

- Store secrets and tokens in a `.env` file adjacent to the `docker-compose.yml` file.
  - The environment variables are automatically interpolated when `docker compose up` is called.
  - If multiple `.env` files are required (for separation of secrets), then use `.env.<CONTAINER_NAME>` (e.g. `.env.gitea`) and add an override in the `docker-compose.yml` file. See the Docker documentation for more details.
- Always used tagged images.
  - Avoid using `latest`. By default, Docker hub automatically tags the most recently pushed image as `latest`, unless overriden by the image maintainer. This means that you might be running bleeding edge/alpha/vulnerable versions.
  - Often images expect a specific version of a container to be running in order for DB migrations to work. This is especially important with Postgres where major versions are not always forwards compatible.
- Separate frontend and backend services using different networks.
  - Docker Compose manages the creation/destruction of these networks for you. Please see the examples below.
- Volumes should be placed in a volumes directory adjacent the `docker-compose.yml` file.
  - Prefix volume directories with the name of the container.
  - See the directory structure below for an example.
- Mount `/etc/timezone` and `/etc/localtime` where timestamps are used by the container.
  - See below for an example.

An example directory structure is shown below:

```
.
├── .env.db
├── .env.gitea
├── .env.tunnel
├── docker-compose.yml
├── start.sh
└── volumes/
```

```
├── gitea_config/
|   └── ...
├── gitea_data/
|   └── ...
└── postgres_data/
    └── ...
```

See Cloudflare Tunnel Configuration for instructions on how to configure a tunnel and get a tunnel token.

Gitea

`.env.db`

```
POSTGRES_USER=gitea
POSTGRES_PASSWORD=gitea
POSTGRES_DB=gitea
```

`.env.gitea`

```
GITEA__database__DB_TYPE=mysql
GITEA__database__HOST=db:3306
GITEA__database__NAME=gitea
GITEA__database__USER=gitea
GITEA__database__PASSWD=gitea
```

`.env.tunnel`

```
TUNNEL_TOKEN=abc...
```

`docker-compose.yml`

```
services:
  tunnel:
    image: cloudflare/cloudflared:2024.4.0       # Use version tags to ensure only stable software is used.
    restart: unless-stopped                      # This restart command helps with crashing services.
    command: tunnel run
    depends_on:
      - gitea                              # Ensure dependencies start in the correct order.
    networks:
      - frontend                           # Use multiple networks to isolate services.
    env_file:
      - .env.tunnel                        # Use environment variables loaded via a .env file for tokens.
  gitea:
    image: gitea/gitea:1.21-rootless
    restart: unless-stopped
    healthcheck:                           # Use healthchecks if possible.
      test: curl --fail http://localhost:3000/api/healthz || exit 1
      interval: 60s
      retries: 5
      start_period: 20s
      timeout: 10s
    depends_on:
      - db
    networks:
      - frontend
      - backend
    volumes:
      - './volumes/gitea_data:/var/lib/gitea'    # Mount volumes into the ./volumes directory.
      - './volumes/gitea_config:/etc/gitea'      # Relative volumes must be wrapped in single quotes.
      - '/etc/timezone:/etc/timezone:ro'         # Mount timezone/localtime so that timestamps are correct.
      - '/etc/localtime:/etc/localtime:ro'
    env_file:
      - .env.gitea
  db:
    image: postgres:14
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready", "-d", "db_prod"]
      interval: 60s
      retries: 5
      start_period: 20s
      timeout: 10s
    networks:
      - backend
    volumes:
      - './volumes/postgres_data:/var/lib/postgresql/data'
    env_file:
      - .env.db
networks:
  backend:
  frontend:
```