# Beginner-Guide: Traefik & Docker Swarm

træfik proxy



HTTP://DOMAIN/FIRST

TRAEFIK.ROUTERS.CONTAINER-1.RULE=PATH(/FIRST)

HTTP://DOMAIN/SECOND

INSTANCE 1

TRAEFIK.ROUTERS.CONTAINER-2.RULE=PATH(/SECOND)

INSTANCE 2

TRAEFIK.ROUTERS.CONTAINER-2.RULE=PATH(/SECOND)
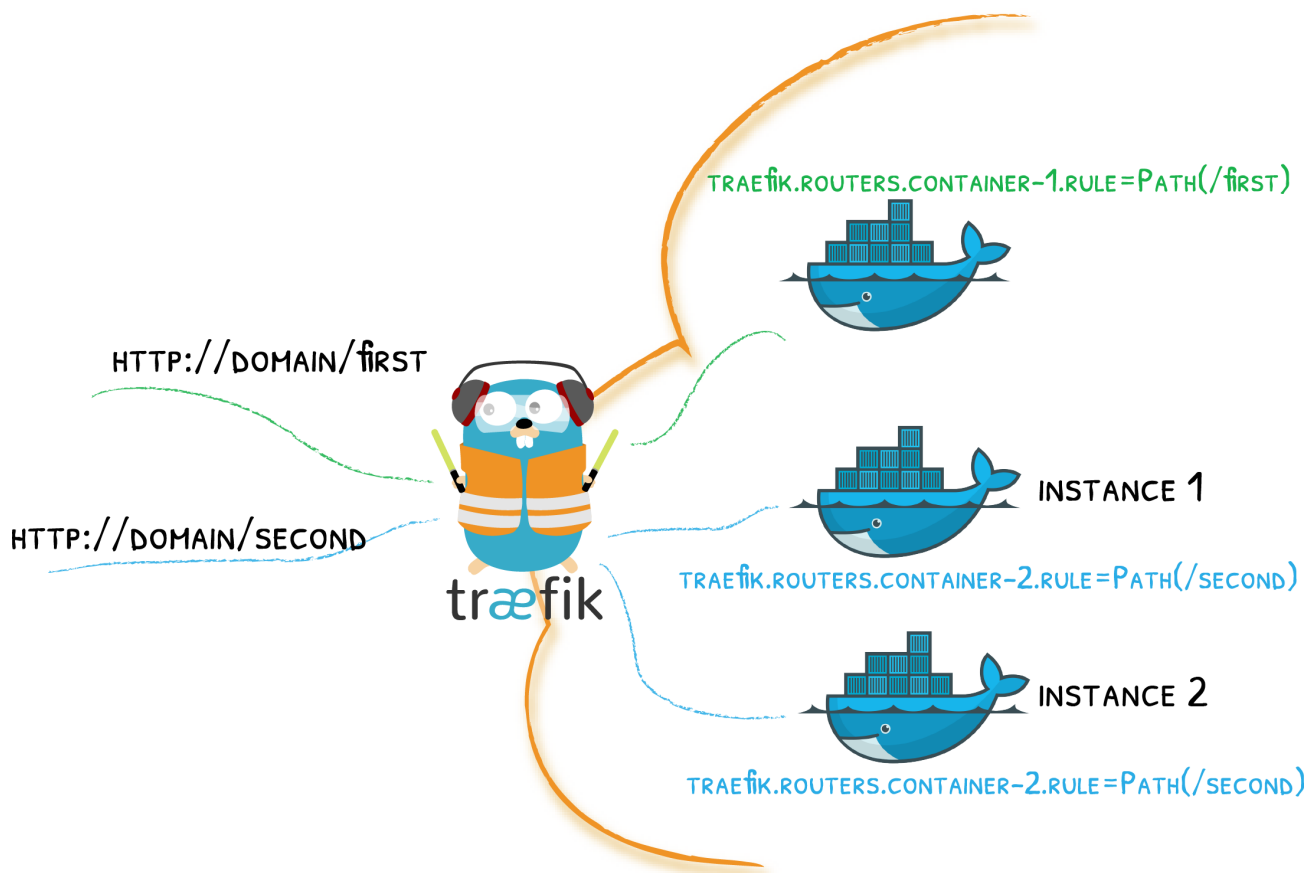
## How Traefik Works with Docker Swarm

Traefik integrates tightly with Docker Swarm, using the **Docker API** to automatically discover services. In **Swarm Mode**, Traefik watches for **service-level labels** rather than container-level labels (which are used in standalone Docker mode). This allows Traefik to dynamically adapt as services are created, updated, or removed.

## Benefits of Using Traefik with Docker Swarm:

- **Dynamic Service Discovery**: Traefik automatically finds and configures services based on Docker labels, which eliminates the need for manually updating configuration files.
- **Scalability**: Traefik automatically adjusts routing as new instances of services are scaled up or down.
- **Real-Time Updates**: As services are added or removed in Swarm, Traefik updates routing rules without needing restarts or manual intervention.

# Example: Deploying Traefik with Docker Swarm

Let's walk through an example where we deploy Traefik as a **reverse proxy** in a Docker Swarm environment. We'll expose a simple web service and configure routing with labels.

## 1. Setting Up Traefik in Docker Swarm

First, we need to deploy Traefik as a service in the Docker Swarm cluster. Traefik requires access to the Docker socket to monitor the containers and services. Here's how to deploy Traefik with Docker Swarm using **Docker Compose**:

```
version: '3.7'
services:
  traefik:
    image: traefik:v3.0
    command:
      - "--api.insecure=true"  # Exposes Traefik's dashboard
      - "--providers.docker=true"  # Enables Docker as the provider
      - "--entrypoints.web.address=:80"  # Defines an HTTP EntryPoint on port 80
    ports:
      - "80:80"      # Expose Traefik's HTTP EntryPoint
      - "8080:8080"   # Expose the Traefik Dashboard
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"  # Access Docker API
    deploy:
      placement:
        constraints:
          - node.role == manager  # Only run on Swarm manager nodes
```

**Explanation**:

- `providers.docker=true` : Enables Traefik to use Docker as a provider.
- `entrypoints.web.address=:80` : Defines an HTTP EntryPoint listening on port 80.
- `api.insecure=true` : Enables Traefik's web dashboard (only for demonstration purposes, not recommended in production).

## 2. **Deploying a Service with Routing Labels**

Next, let's deploy a simple web service, such as Nginx, and attach routing labels so that Traefik can automatically route traffic to it.

```
version: '3.7'
services:
  nginx:
    image: nginx
    deploy:
      labels:
        - "traefik.enable=true"
        - "traefik.http.routers.nginx.rule=Host(`nginx.local`)"
        - "traefik.http.services.nginx.loadbalancer.server.port=80"
```

**Explanation**:

- `traefik.enable=true` : Tells Traefik to expose this service.
- `traefik.http.routers.nginx.rule=Host('nginx.local')` : Defines a routing rule that routes requests with the `Host` header `nginx.local` to this service.
- `traefik.http.services.nginx.loadbalancer.server.port=80` : Specifies the internal port for Nginx.

> Now, you can deploy this configuration in your Swarm cluster with:
> `docker stack deploy -c docker-compose.yml my-stack`
> Or you can run the Stack in Portainer :)

# Understanding Key Configuration Elements

## Port Detection in Docker Swarm

Traefik does not automatically detect ports in Docker Swarm mode. You must explicitly set the **port label** (`traefik.http.services.<service_name>.loadbalancer.server.port`) to tell Traefik which port to use for the service. This ensures that Traefik can properly route requests to the service.

## Host Networking with Traefik

If you are exposing containers configured with **host networking**, Traefik resolves the host IP based on the following priorities:

1. `host.docker.internal`
2. `host.containers.internal`
3. If both fail, it falls back to `127.0.0.1`.

## IPv4 and IPv6

By default, Traefik prioritizes the **IPv4** address of a container, even in an **IPv6**-enabled Docker stack. If you want Traefik to use the IPv6 address, make sure your Docker configuration supports IPv6 routing.

---

# Scheduling Traefik on Swarm Nodes

In Docker Swarm mode, only **manager nodes** can access the Docker Swarm API, which Traefik needs to dynamically discover services. Therefore, you must schedule Traefik on **manager nodes**. Here's how to do it:

**Example for Docker Compose:**

```
services:
  traefik:
    image: traefik:v3.0
    deploy:
```

```
      placement:
       constraints:
         - "node.role == manager"
```

# Docker API Access and Security Considerations

Since Traefik requires access to the Docker API via the **docker.sock** socket, this raises potential security risks. Anyone with access to Traefik can also gain access to the Docker API and, by extension, the underlying host. To mitigate this risk:

- Only expose the Docker socket to trusted services.
- Consider securing the Docker socket with a **TLS connection**.

Here's an example of a secure Docker API configuration with **TLS**:

```
providers:
  swarm:
    tls:
      cert: "/path/to/cert.crt"
      key: "/path/to/key.key"
      ca: "/path/to/ca.crt"
```

> This configuration ensures that the Docker API connection is secured using TLS.

# Conclusion

Traefik's integration with Docker Swarm provides a powerful, dynamic solution for managing services at scale. By using labels on services, Traefik can automatically discover, configure, and route traffic without the need for manual intervention. It's particularly useful for dynamic environments where services are frequently added or removed, as Traefik handles these changes in real time.

This beginner-friendly overview of Traefik with Docker Swarm should give you a solid foundation for deploying and routing your services dynamically. As you become more familiar, you can explore advanced configurations such as load balancing strategies, security features, and using external providers like Kubernetes.