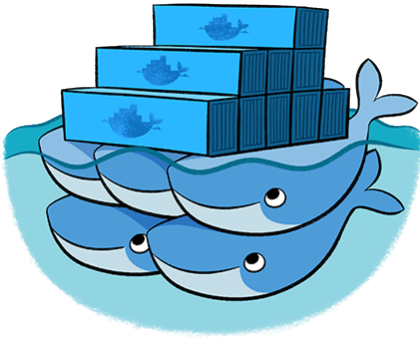


Understanding Docker Swarm Mode Routing Mesh



Docker Swarm Mode introduces the concept of a **routing**

mesh, a powerful feature that simplifies the process of exposing services to external clients in a swarm cluster. This tutorial provides an overview of how the **ingress network** and routing mesh work, how to publish ports for services, and how to configure an external load balancer like HAProxy to interact with a Swarm service.

What is the Routing Mesh?

The **routing mesh** in Docker Swarm Mode allows all nodes in the swarm to accept connections to a published service port, even if the service isn't running on that particular node. The routing mesh automatically routes incoming requests to a node where an instance of the service is active, ensuring the requests are handled efficiently and transparently.

Ingress Network

The **ingress network** is a special network created by Docker for the purpose of handling the routing mesh. All Swarm services are connected to the ingress network, allowing them to communicate and route requests to the correct node.

Key Ports to Open

Before you enable Swarm Mode and use the ingress network, make sure the following ports are open between the swarm nodes:

- **Port 7946 (TCP/UDP):** For container network discovery.
 - **Port 4789 (UDP):** For the container ingress network (VXLAN).
-

How the Routing Mesh Works

When you publish a service in Docker Swarm, each node in the swarm can receive traffic for that service, regardless of whether the service is running on that node or not. The swarm manager's routing mesh ensures that requests are forwarded to the appropriate node where the service is active.

Example:

You have three nodes in your swarm:

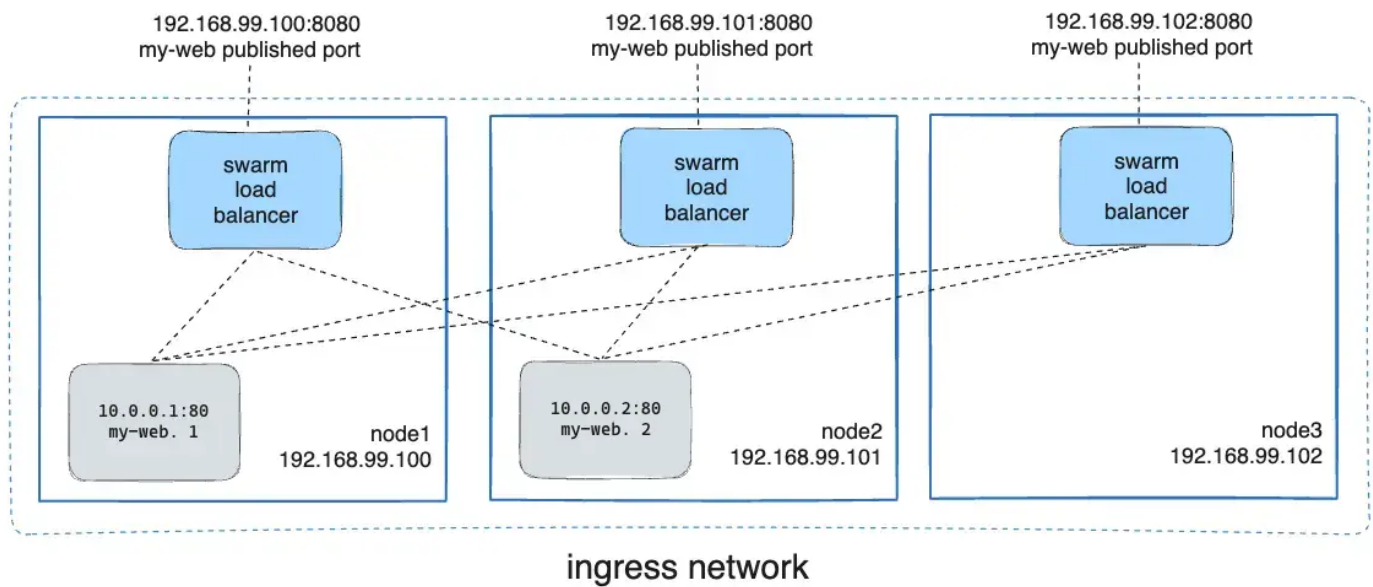
- node1 with IP 192.168.99.100
- node2 with IP 192.168.99.101
- node3 with IP 192.168.99.102

You deploy a service called my-web using the Nginx container. Even if the Nginx container is only running on node1 and node2, you can send a request to node3, and Docker will route that request to one of the active containers running on node1 or node2.

Here's an example of the command to create the service with two replicas:

```
docker service create \
  --name my-web \
  --publish published=8080,target=80 \
  --replicas 2 \
  nginx
```

In this case, the Nginx service is available on port 8080 on all swarm nodes.



In the diagram above, you can see that requests sent to any node on port 8080 are routed to the appropriate Nginx instance.

Publishing a Port for a Service

To expose a service to the outside world, you use the `--publish` flag when creating the service. This flag lets you map a port on the node (published port) to a port inside the container (target port).

For example:

```
docker service create \
  --name my-web \
  --publish published=8080,target=80 \
  nginx
```

This command publishes port `8080` on the swarm nodes and maps it to port `80` inside the Nginx container.

- **published:** The port that the swarm makes available outside the container.
- **target:** The port that the container listens on (inside the container).

Viewing Published Ports with `docker service inspect`

You can use `docker service inspect` to view detailed information about a service, including which ports have been published.

For example, to inspect the `my-web` service:

```
docker service inspect --format="{{json .Endpoint.Spec.Ports}}" my-web
```

The output will show the `TargetPort` (the container's internal port) and the `PublishedPort` (the port on the swarm nodes):

```
[{"Protocol":"tcp","TargetPort":80,"PublishedPort":8080}]
```

Configuring an External Load Balancer with the Routing Mesh

In a real-world production scenario, you may want to use an **external load balancer** such as HAProxy to handle traffic across multiple swarm nodes. The load balancer can distribute incoming traffic to the nodes in the swarm, which will then use the routing mesh to route the traffic to the correct container.

Example HAProxy Configuration

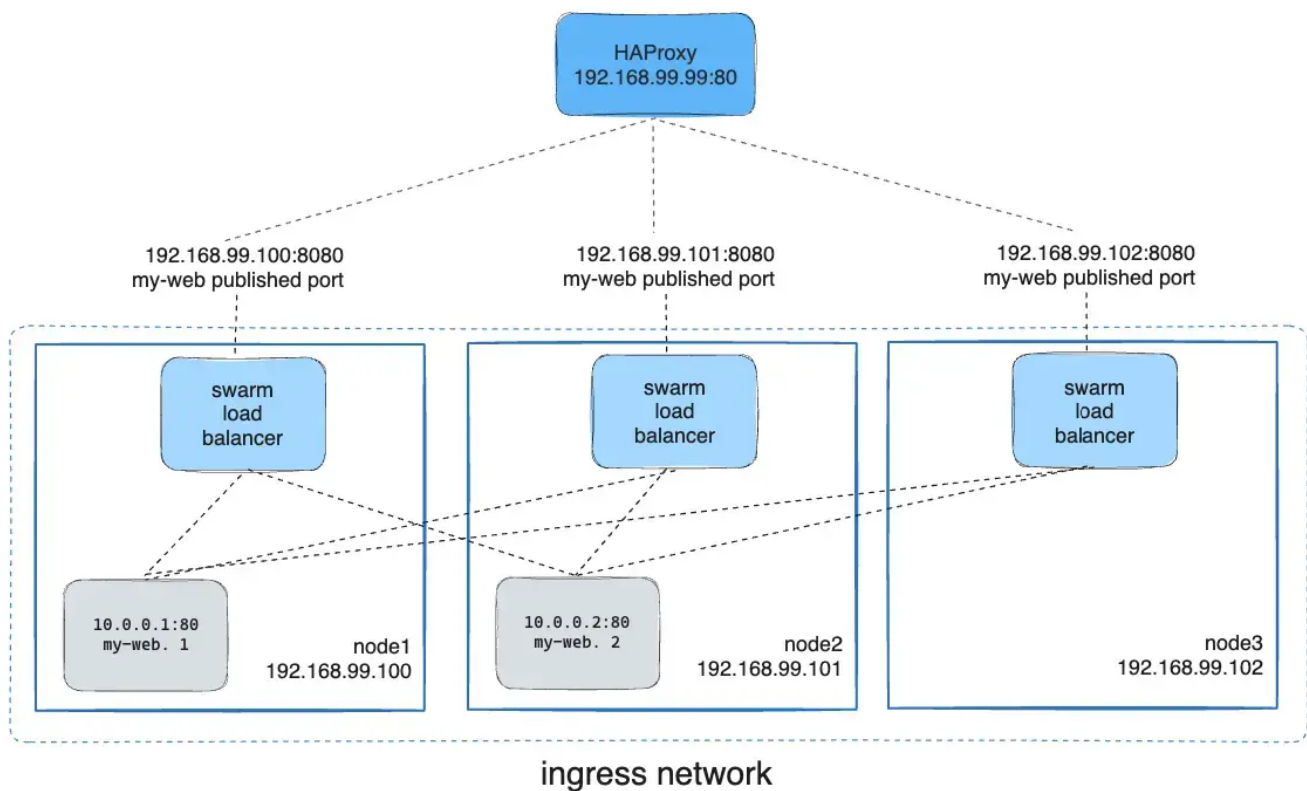
The following HAProxy configuration listens on port `80` and forwards requests to the `my-web` service running on port `8080` on the swarm nodes:

```
global
    log /dev/log local0
    log /dev/log local1 notice

frontend http_front
```

```
backend http_back
    balance roundrobin
    server node1 192.168.99.100:8080 check
    server node2 192.168.99.101:8080 check
    server node3 192.168.99.102:8080 check
```

This configuration ensures that incoming traffic on port 80 is distributed across the nodes (node1 , node2 , and node3) on port 8080 .



In the above diagram, the HAProxy load balancer distributes requests across all nodes in the swarm, and the swarm routing mesh ensures that traffic is forwarded to an active container.

Bypassing the Routing Mesh (Host Mode)

In some cases, you may want to bypass the routing mesh so that requests are sent directly to the node that is running the service. This is useful when you want to ensure that only nodes running the service are accessible on the published port. This mode is referred to as **host mode**.

To bypass the routing mesh, use the `--publish` flag with the `mode=host` option:

```
docker service create --name dns-cache \  
  --publish published=53,target=53,protocol=udp,mode=host \  
  --mode global \  
  dns-cache
```

In host mode, traffic sent to a node will only be handled if that node is running the service task. Otherwise, the connection will fail.

Using an External Load Balancer Without the Routing Mesh

If you want to bypass the routing mesh entirely, you can configure an external load balancer to handle traffic without relying on the Swarm's built-in load balancing.

Use `--endpoint-mode dnsrr` to configure Docker to return a list of IP addresses for the nodes running the service when queried, rather than a virtual IP. This allows your external load balancer to directly handle traffic distribution based on DNS entries.

Example:

```
docker service create \  
  --name my-web \  
  --publish published=8080,target=80 \  
  --endpoint-mode dnsrr \  
  nginx
```

In this mode, the load balancer directly routes traffic to nodes running the service, without going through the swarm routing mesh.

Conclusion

Docker Swarm Mode's routing mesh and ingress network provide powerful and flexible ways to expose services to external clients. By using the routing mesh, all swarm nodes can participate in traffic routing, providing high availability and fault tolerance. For more control, you can configure external load balancers like HAProxy or bypass the routing mesh entirely to meet specific needs.

Whether you are using the default routing mesh or integrating with an external load balancer, Docker Swarm Mode simplifies the process of deploying and scaling services across distributed systems.

Revision #1

Created 11 September 2024 14:32:42 by aeoneros

Updated 11 September 2024 14:39:56 by aeoneros