**Issue**

Typically, container-container networking issues manifest themselves as intermittent connection timeouts. They could also manifest as a complete failure of container-container networking.

This walkthrough will guide you to collect the necessary information to determine the root cause of a network connectivity problem.

**Prerequisites**

First, determine the names of the affected Docker swarm services, networks, and hosts.

Identify a Docker host from which network requests are not answered correctly. If the requests are submitted through the ingress network, then to a frontend service, then via another network to a backend service, then start troubleshooting by splitting the problem in half and using `netshoot` to connect from the frontend directly to the backend service.

**Resolution**

To troubleshoot container-container networking issues, start by examining the user-defined network. If the problem isn't found, move to troubleshooting the ingress network. Both are discussed below.

**Troubleshooting a User-Defined Network**

1. Start a `netshoot` container reusing the network namespace of the affected "frontend" container:

   ```
   docker run -it --rm --net container:<^><container_name><^^> nicolaka/netshoot
   ```

2. Look up all backend IP addresses:

   ```
   nslookup tasks.backend_service_name 2>/dev/null | awk '$1=="Address"{print $3}'
   ```

3. Issue a netcat TCP test to each backend task IP address on the port where it should be listening:

   ```
   for backend_ip in $(nslookup tasks.<^><backend_service_name><^^> 2>/dev/null | awk '$1=="Address" {print $3}'); do
     nc -zw2 $backend_ip <^><listening_port><^^> &>/dev/null || echo $backend_ip failed;
   done
   ```

   **Where:**

   - `<backend_service_name>` is changed to the backend service name
   - `<listening_port>` is changed to the port the backend tasks are listening on

4. If no connections fail but requests submitted via the ingress network continue to have problems, move to the next section on troubleshooting the ingress network. If no connections fail, and the issue has only been seen container-container, then check another set of services or hosts until one task fails to reach another.
5. For any backend IP addresses reported as failed, do a reverse name lookup of the affected IP to determine its service name and task id:

```
nslookup <^><IP_address><^^>
```

Results will be formatted `<^>service_name.slot.taskid.network_name<^^>`

6. Exit the netshoot container and collect `docker network inspect -v` against the network between the two containers. Note the HostIP of tasks in the Services section that failed the netcat test.

7. On a manager, for the set of all failed service names and tasks, collect the following:

```
docker service ps <^><service_name><^^>
docker inspect --type task <^><task_id><^^>
```

8. For tasks that are still present (`inspect --type task` task returns output), note their Created and Updated times. Collect Docker daemon logs covering this time frame from the netshoot host, all managers, and hosts of unresponsive tasks as identified by their HostIP.

If your Docker daemon logs to journald, for example:

```
journalctl -u docker --no-pager --since "YYYY-MM-DD 00:00" --until "YYYY-MM-DD 00:00"
```

9. Collect the output of [https://github.com/adamancini/libnetwork/blob/improved_support_script/support.sh](https://github.com/adamancini/libnetwork/blob/improved_support_script/support.sh) from all hosts in the network path. This will expose and allow for the verification of the kernel programming if needed.

## Troubleshooting the Ingress Network

Service discovery is disabled on the ingress network for security reasons, so you can’t use `nslookup tasks.service` to establish which backend IPs to test. Instead use the ipvs loadbalancer programming of the kernel.

1. On a manager, use `docker service inspect` to identify the VIP for the service on the ingress network (where `<^><service_name><^^>` is changed to the name of the service):

```
ingress_id=$(docker network ls -qf name=ingress --no-trunc); docker service inspect <^>
<service_name><^^> --format '{{range .Endpoint.VirtualIPs}}{{if eq "$ingress_id" .NetworkID}}{{.Addr}}{{end}}{{end}}{{end}}'
```

2. Using `curl` identify a Docker host from which network requests to the ingress published port of the service are not answered correctly:

```
curl -4 -H 'Host: myapp.example.com' http://host:<^><listening_port><^^>
```

3. Use a netshoot container to enter into the ingress_sbox namespace on that host:

```
docker run -it --rm -v /var/run/docker/netns:/netns --privileged=true nicolaka/netshoot nsenter --net=/netns/ingress_sbox sh
```

4. Look up the decimal ingress firewall mark for the service in question:
iptables -n -t mangle | awk '/<vip>/<vip>/' {printf("%d", $NF)}'; echo

Where <vip> is replaced with the IP address of the service determined by service inspect.

5. List all the backend task IP addresses using ipvsadm

ipvsadm -l -f <fwmark>

Where <fwmark> is the decimal firewall mark extracted from the iptables mangle table.

6. Issue a netcat TCP test to each task IP address on the port where it should be listening:

```bash
for backend_ip in $(ipvsadm -l -f <fwmark> | awk 'NR>3{print $2}' | cut -d: -f1); do
    nc -zw1 $backend_ip <listening_port> &>/dev/null || echo $backend_ip failed;
done
```

Where:
* <listening_port> is changed to the port the backend tasks are listening on
* <fwmark> is the changed to the decimal firewall mark extracted from the iptables mangle table

7. Exit the netshoot container and collect docker network inspect -v ingress. Note the HostIP, service name, and task ID of any tasks with failed backend IP addresses.

8. On a manager, for the set of all failed service names and tasks, collect the following:

```bash
docker service ps <service_name>
docker inspect --type task <task_id>
```

9. For tasks that are still present (inspect --type task returns output), note their Created and Updated times. Collect Docker daemon logs covering this time frame from the netshoot host, all managers, and hosts of unresponsive tasks as identified by their HostIP.

If your Docker daemon logs to journald, for example:

```bash
journalctl -u docker --no-pager --since "YYYY-MM-DD 00:00" --until "YYYY-MM-DD 00:00"
```

10. Collect the output of https://github.com/docker/libnetwork/blob/master/support/support.sh from all hosts in the network path. This will expose and allow us to verify kernel programming if needed.