

AI-03084 Docker Network

Owner	Ⓜ Rigel Arcayan
Tags	Project
Created time	@October 16, 2025 11:49 AM

Delve on docker networks. How can I inspect what is inside the docker network
Excellent question — understanding Docker networks deeply is key to mastering inter-container communication, isolation, and debugging connectivity issues.

Let's go through this systematically:

1. Concept Overview

Docker networks provide virtual isolated environments where containers communicate using IP addresses, DNS names, or exposed ports.

Main network types:

- bridge – default; local container-to-container communication on the same host
- host – shares the host's network namespace (no isolation)
- macvlan – containers appear as physical devices on your LAN
- overlay – used in Swarm for multi-host networking
- none – completely isolated (no network stack)

2. List All Docker Networks

docker network ls

Example output:

```
NETWORK ID   NAME           DRIVER  SCOPE
3f5b2b9a3c90 bridge        bridge  local
e4d1b94b7a1b host          host    local
```

```
b9d3ef2c3a01 none null local
a8e2bfa6dc9f dockerlab_net bridge local
```

3. Inspect a Specific Network

```
docker network inspect dockerlab_net
```

This shows:

- Network name, ID, driver, scope
- Subnet and gateway
- Connected containers with their:
 - Container name
 - IPv4/IPv6 address
 - MAC address
 - Endpoint ID
-

Sample output snippet:

```
[
{
  "Name": "dockerlab_net",
  "Id": "a8e2bfa6dc9f",
  "Driver": "bridge",
  "IPAM": {
    "Config": [
      {
        "Subnet": "172.20.0.0/16",
        "Gateway": "172.20.0.1"
      }
    ]
  }
}
```

```
]
},
"Containers": {
  "0f5a8b": {
    "Name": "wordpress",
    "IPv4Address": "172.20.0.3/16",
    "MacAddress": "02:42:ac:14:00:03"
  },
  "8a3cd0": {
    "Name": "mariadb",
    "IPv4Address": "172.20.0.2/16",
    "MacAddress": "02:42:ac:14:00:02"
  }
}
}
```

4. List Containers Attached to a Network

You can filter with:

```
docker network inspect -f '{{range .Containers}}{{.Name}} {{end}}' dockerlab_net
```

Output:

```
wordpress mariadb phpmyadmin
```

Or more detailed (with IPs):

```
docker network inspect -f '{{range .Containers}}{{.Name}}: {{.IPv4Address}}
{{end}}' dockerlab_net
```

5. Connect or Disconnect Containers

- Connect an existing container:

```
docker network connect dockerlab_net wordpress
```

-

- Disconnect a container:

```
docker network disconnect dockerlab_net wordpress
```

6. Dive Inside a Network (Advanced)

You can use a temporary network inspector container:

```
docker run -it --rm --network dockerlab_net alpine sh
```

Inside Alpine, try:

```
ping mariadb
```

```
ping wordpress
```

If DNS is working, you'll see responses — confirming the containers “see” each other.

7. Check Network Interfaces on the Host

Each bridge network creates a virtual bridge interface on the host.

```
ip link show
```

Look for interfaces like br-xxxxxxxxxxxxx.

Inspect IP/subnet:

```
ip addr show br-xxxxxxxxxxxxx
```

8. Visualize Network Topology (Optional)

You can use Portainer or Docker Desktop (if available) to visualize container interconnections under the “Networks” tab.

9. Summary (Quick Commands)

Purpose	Command
List networks	<code>docker network ls</code>
Inspect details	<code>docker network inspect <name></code>
List containers in a network	<code>docker network inspect -f '{{range .Containers}}{{.Name}}\n{{end}}' <name></code>
Connect container	<code>docker network connect <network> <container></code>
Disconnect container	<code>docker network disconnect <network> <container></code>
Run debug shell inside network	<code>docker run -it --rm --network <network> alpine sh</code>

Would you like me to add a “Docker Network Deep Dive Map” (visual diagram showing bridge, macvlan, host, overlay, and how containers communicate)?

Yes, Rigel — you can edit a Docker network, but the extent of what you can change depends on the network type and what you’re trying to modify. Here’s a breakdown tailored to your protocol mindset:

Docker Network Editing: What’s Possible

1. Inspect First

Use this to understand the current configuration:

```
docker network inspect <network_name>
```

1. Limited Live Edits

Docker doesn’t allow full live editing of networks. However, you can do the following:


- Connect/disconnect containers:
`docker network connect <network_name> <container_name>`
`docker network disconnect <network_name> <container_name>`

- Change container IP (on user-defined bridge): You must disconnect and reconnect with `-ip`: `docker network disconnect <network_name> <container_name>`
`docker network connect --ip <new_ip> <network_name> <container_name>`
- Add aliases: `docker network connect --alias <alias_name> <network_name> <container_name>`

1. Recreate for Deep Changes

To change subnet, gateway, or driver, you must remove and recreate the network:

```
docker network rm <network_name>
docker network create \
--driver bridge \
--subnet 192.168.100.0/24 \
--gateway 192.168.100.1 \
<new_network_name>
```

 Tip: Use `docker-compose` or modular `docker-compose.override.yml` to define networks declaratively for easier versioning and protocol gifting.

Protocol Design Suggestion

For your DockerLab Part 2, consider creating a network manifest in your Codex archive:

- `network_name`
 - `driver`
 - `subnet/gateway`
 - `attached_services`
 - `aliases`
 - `compose linkage`

Would you like help drafting a reusable network protocol template for your mentoring archive?