Networks top-level elements

Table of contents

Networks let services communicate with each other. By default Compose sets up a single network for your app. Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by the service's name. The top-level `networks` element lets you configure named networks that can be reused across multiple services.

To use a network across multiple services, you must explicitly grant each service access by using the networks attribute within the `services` top-level element. The `networks` top-level element has additional syntax that provides more granular control.

Examples

Basic example

In the following example, at runtime, networks `front-tier` and `back-tier` are created and the `frontend` service is connected to `front-tier` and `back-tier` networks.

```
services:
 frontend:
   image: example/webapp
   networks:
     - front-tier
     - back-tier

networks:
 front-tier:
 back-tier:
```

Advanced example

```
services:
 proxy:
   build: ./proxy
   networks:
     - frontend
 app:
   build: ./app
   networks:
     - frontend
     - backend
 db:
   image: postgres
   networks:
     - backend

networks:
 frontend:
   # Use a custom driver
   driver: custom-driver-1
 backend:
   # Use a custom driver which takes special options
   driver: custom-driver-2
   driver_opts:
     foo: "1"
     bar: "2"
```

The advanced example shows a Compose file which defines two custom networks. The `proxy` service is isolated from the `db` service, because they do not share a network in common. Only `app` can talk to both.

Attributes

driver

`driver` specifies which driver should be used for this network. Compose returns an error if the driver is not available on the platform.

```
networks:
 db-data:
   driver: bridge
```

For more information on drivers and available options, see Network drivers.

driver_opts

`driver_opts` specifies a list of options as key-value pairs to pass to the driver. These options are driver-dependent. Consult the driver's documentation for more information.

```
networks:
 db-data:
   driver_opts:
     foo: "bar"
     baz: 1
```

attachable

If `attachable` is set to `true`, then standalone containers should be able to attach to this network, in addition to services. If a standalone container attaches to the network, it can communicate with services and other standalone containers that are also attached to the network.

```
networks:
 mynet1:
   driver: overlay
   attachable: true
```

enable_ipv6

`enable_ipv6` enables IPv6 networking. For an example, see step four of Create an IPv6 network.

external

If set to `true`:

- `external` specifies that this networkâ■■s lifecycle is maintained outside of that of the application. Compose doesn't attempt to create these networks, and returns an error if one doesn't exist.
- All other attributes apart from name are irrelevant. If Compose detects any other attribute, it rejects the Compose file as invalid.

In the example below, `proxy` is the gateway to the outside world. Instead of attempting to create a network, Compose queries the platform for an existing network simply called `outside` and connects the `proxy` service's containers to it.

```
services:
 proxy:
   image: example/proxy
   networks:
     - outside
     - default
 app:
   image: example/app
   networks:
     - default

networks:
 outside:
   external: true
```

ipam

`ipam` specifies a custom IPAM configuration. This is an object with several properties, each of which is optional:

- `driver`: Custom IPAM driver, instead of the default.
  `config`: A list with zero or more configuration elements, each containing a:
  - `subnet`: Subnet in CIDR format that represents a network segment
  - `ip_range`: Range of IPs from which to allocate container IPs
  - `gateway`: IPv4 or IPv6 gateway for the master subnet
  - `aux_addresses`: Auxiliary IPv4 or IPv6 addresses used by Network driver, as a mapping from hostname to IP
- `options`: Driver-specific options as a key-value mapping.

```
networks:
 mynet1:
   ipam:
     driver: default
     config:
       - subnet: 172.28.0.0/16
         ip_range: 172.28.5.0/24
         gateway: 172.28.5.254
         aux_addresses:
           host1: 172.28.1.5
           host2: 172.28.1.6
           host3: 172.28.1.7
     options:
       foo: bar
       baz: "0"
```

internal

By default, Compose provides external connectivity to networks. `internal`, when set to `true`, allows you to create an externally isolated network.

labels

Add metadata to containers using `labels`. You can use either an array or a dictionary.

It is recommended that you use reverse-DNS notation to prevent labels from conflicting with those used by other software.

```
networks:
 mynet1:
   labels:
     com.example.description: "Financial transaction network"
     com.example.department: "Finance"
     com.example.label-with-empty-value: ""
```

```
networks:
 mynet1:
   labels:
     - "com.example.description=Financial transaction network"
     - "com.example.department=Finance"
     - "com.example.label-with-empty-value"
```

Compose sets `com.docker.compose.project` and `com.docker.compose.network` labels.

name

`name` sets a custom name for the network. The name field can be used to reference networks which contain special characters. The name is used as is and is not scoped with the project name.

```
networks:
 network1:
   name: my-app-net
```

It can also be used in conjunction with the `external` property to define the platform network that Compose should retrieve, typically by using a parameter so the Compose file doesn't need to hard-code runtime specific values:

```
networks:
 network1:
   external: true
   name: "${NETWORK_ID}"
```

Additional resources

For more examples, see Networking in Compose.