11.03.2024

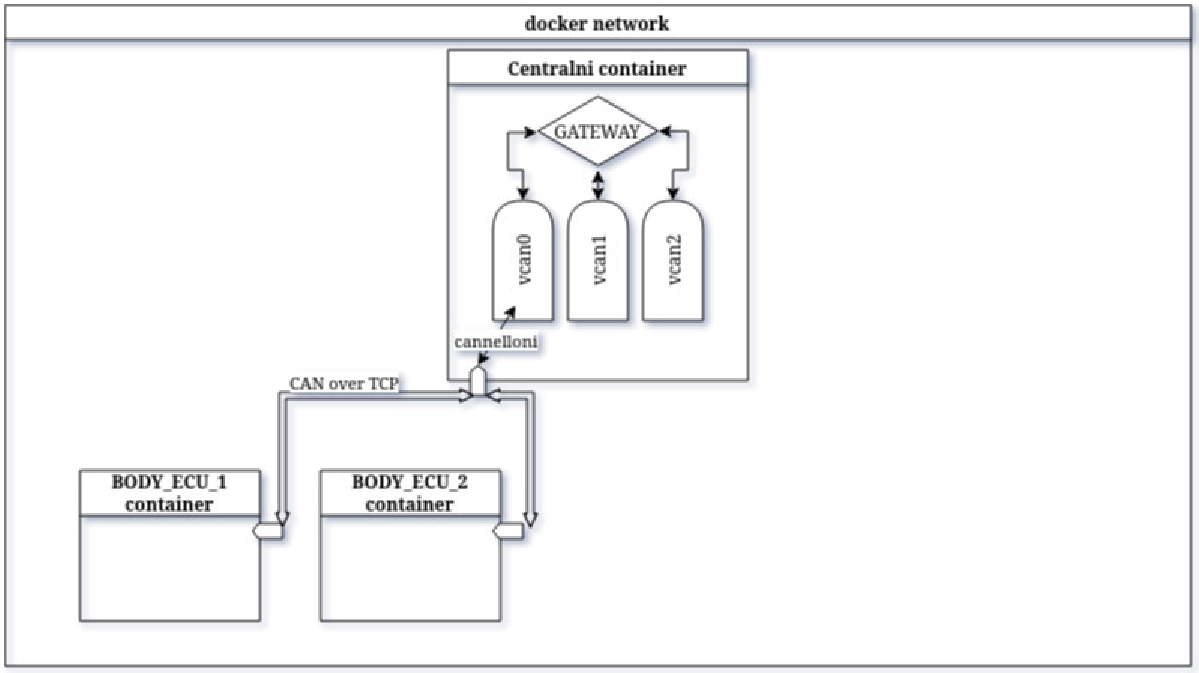
Kljucno

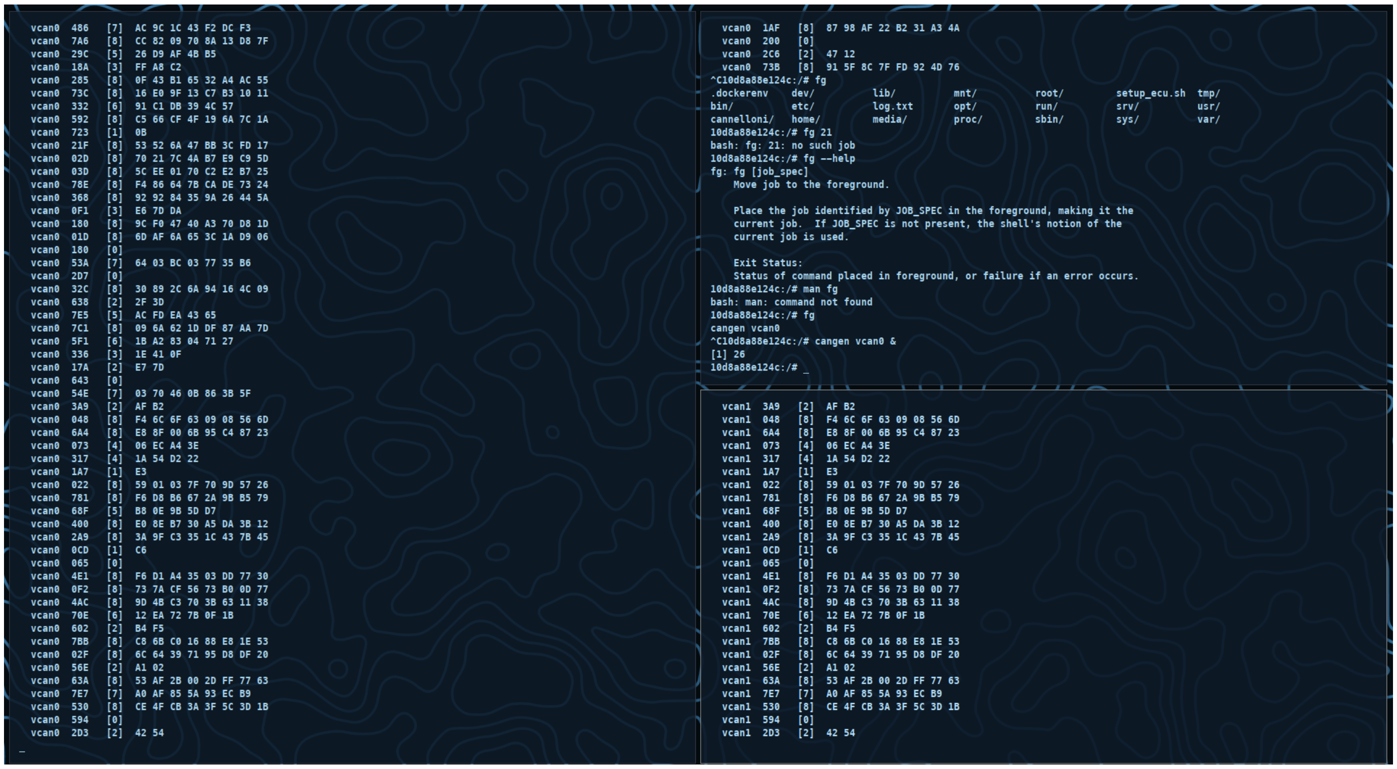
- neovisno o tome koristi li se can network driver za docker, ecu-ovi trebaju biti spojeni i u docker bridge mrezu kako bi mogli komunicirati s modularnim gui komponentama (primjerice prikaz brzine)

Issue za cannelloni tcp server koji prihvaca vise klijenata

- https://github.com/mguentner/cannelloni/issues/17
  - predlozeno rjesenje je pokrenut vise istanci

Cannelloni komunikacija izmedju dva ecua preko jednog vcan sucelja u gw containeru koji pokrece vise cannelloni instanci na razlicitim portovima.





```
  vcan0  486   [7]  AC 9C 1C 43 F2 DC F3
  vcan0  7A6   [8]  CC 82 09 70 8A 13 D8 7F
  vcan0  29C   [5]  26 D9 AF 4B B5
  vcan0  18A   [3]  FF A8 C2
  vcan0  285   [8]  0F 43 B1 65 32 A4 AC 55
  vcan0  73C   [8]  16 E0 9F 13 C7 B3 10 11
  vcan0  332   [6]  91 C1 DB 39 4C 57
  vcan0  592   [8]  C5 66 CF 4F 19 6A 7C 1A
  vcan0  723   [1]  0B
  vcan0  21F   [8]  53 52 6A 47 BB 3C FD 17
  vcan0  02D   [8]  70 21 7C 4A B7 E9 C9 5D
  vcan0  03D   [8]  5C EE 01 70 C2 E2 B7 25
  vcan0  78E   [8]  F4 86 64 7B CA DE 73 24
  vcan0  368   [8]  92 92 84 35 9A 26 44 5A
  vcan0  0F1   [3]  E6 7D DA
  vcan0  180   [8]  9C F0 47 40 A3 70 D8 1D
  vcan0  01D   [8]  6D AF 6A 65 3C 1A D9 06
  vcan0  180   [0]
  vcan0  53A   [7]  64 03 BC 03 77 35 B6
  vcan0  2D7   [0]
  vcan0  32C   [8]  30 89 2C 6A 94 16 4C 09
  vcan0  638   [2]  2F 3D
  vcan0  7E5   [5]  AC FD EA 43 65
  vcan0  7C1   [8]  09 6A 62 1D DF 87 AA 7D
  vcan0  5F1   [6]  1B A2 83 04 71 27
  vcan0  336   [3]  1E 41 0F
  vcan0  17A   [2]  E7 7D
  vcan0  643   [0]
  vcan0  54E   [7]  03 70 46 0B 86 3B 5F
  vcan0  3A9   [2]  AF B2
  vcan0  048   [8]  F4 6C 6F 63 09 08 56 6D
  vcan0  6A4   [8]  E8 8F 00 6B 95 C4 87 23
  vcan0  073   [4]  06 EC A4 3E
  vcan0  317   [4]  1A 54 D2 22
  vcan0  1A7   [1]  E3
  vcan0  022   [8]  59 01 03 7F 70 9D 57 26
  vcan0  781   [8]  F6 D8 B6 67 2A 9B B5 79
  vcan0  68F   [5]  B8 0E 9B 5D D7
  vcan0  400   [8]  E0 8E B7 30 A5 DA 3B 12
  vcan0  2A9   [8]  3A 9F C3 35 1C 43 7B 45
  vcan0  0CD   [1]  C6
  vcan0  065   [0]
  vcan0  4E1   [8]  F6 D1 A4 35 03 DD 77 30
  vcan0  0F2   [8]  73 7A CF 56 73 B0 0D 77
  vcan0  4AC   [8]  9D 4B C3 70 3B 63 11 38
  vcan0  70E   [6]  12 EA 72 7B 0F 1B
  vcan0  602   [2]  B4 F5
  vcan0  7BB   [8]  C8 6B C0 16 88 E8 1E 53
  vcan0  02F   [8]  6C 64 39 71 95 D8 DF 20
  vcan0  56E   [2]  A1 02
  vcan0  63A   [8]  53 AF 2B 00 2D FF 77 63
  vcan0  7E7   [7]  A0 AF 85 5A 93 EC B9
  vcan0  530   [8]  CE 4F CB 3A 3F 5C 3D 1B
  vcan0  594   [0]
  vcan0  2D3   [2]  42 54
_
```

```
  vcan0  1AF   [8]  87 98 AF 22 B2 31 A3 4A
  vcan0  200   [0]
  vcan0  2C6   [2]  47 12
  vcan0  73B   [8]  91 5F 8C 7F FD 92 4D 76
^C10d8a88e124c:/# fg
.dockerenv    dev/          lib/          mnt/          root/         setup_ecu.sh  tmp/
bin/          etc/          log.txt       opt/          run/          srv/          usr/
cannelloni/   home/         media/        proc/         sbin/         sys/          var/
10d8a88e124c:/# fg 21
bash: fg: 21: no such job
10d8a88e124c:/# fg --help
fg: fg [job_spec]
    Move job to the foreground.

    Place the job identified by JOB_SPEC in the foreground, making it the
    current job.  If JOB_SPEC is not present, the shell's notion of the
    current job is used.

    Exit Status:
    Status of command placed in foreground, or failure if an error occurs.
10d8a88e124c:/# man fg
bash: man: command not found
10d8a88e124c:/# fg
cangen vcan0
^C10d8a88e124c:/# cangen vcan0 &
[1] 26
10d8a88e124c:/# _
```

```
  vcan1  3A9   [2]  AF B2
  vcan1  048   [8]  F4 6C 6F 63 09 08 56 6D
  vcan1  6A4   [8]  E8 8F 00 6B 95 C4 87 23
  vcan1  073   [4]  06 EC A4 3E
  vcan1  317   [4]  1A 54 D2 22
  vcan1  1A7   [1]  E3
  vcan1  022   [8]  59 01 03 7F 70 9D 57 26
  vcan1  781   [8]  F6 D8 B6 67 2A 9B B5 79
  vcan1  68F   [5]  B8 0E 9B 5D D7
  vcan1  400   [8]  E0 8E B7 30 A5 DA 3B 12
  vcan1  2A9   [8]  3A 9F C3 35 1C 43 7B 45
  vcan1  0CD   [1]  C6
  vcan1  065   [0]
  vcan1  4E1   [8]  F6 D1 A4 35 03 DD 77 30
  vcan1  0F2   [8]  73 7A CF 56 73 B0 0D 77
  vcan1  4AC   [8]  9D 4B C3 70 3B 63 11 38
  vcan1  70E   [6]  12 EA 72 7B 0F 1B
  vcan1  602   [2]  B4 F5
  vcan1  7BB   [8]  C8 6B C0 16 88 E8 1E 53
  vcan1  02F   [8]  6C 64 39 71 95 D8 DF 20
  vcan1  56E   [2]  A1 02
  vcan1  63A   [8]  53 AF 2B 00 2D FF 77 63
  vcan1  7E7   [7]  A0 AF 85 5A 93 EC B9
  vcan1  530   [8]  CE 4F CB 3A 3F 5C 3D 1B
  vcan1  594   [0]
  vcan1  2D3   [2]  42 54
```

3 ecu-a na dvije sabirnice, gw procesi:

```
8c11062d772b:/# ps -a
PID   USER     TIME  COMMAND
    1 root      0:00 {setup_gw.sh} /bin/bash /setup_gw.sh
   13 root      0:00 /bin/bash
   17 root      0:00 cannelloni/cannelloni -I vcan1 -C s -R ecu1 -r 10001 -l 10001 -f
   22 root      0:00 cannelloni/cannelloni -I vcan2 -C s -R ecu2 -r 10002 -l 10002 -f
   46 root      0:00 cannelloni/cannelloni -I vcan1 -C s -R ecu3 -r 10003 -l 10003 -f
   49 root      0:00 ps -a
```

gw cangw pravila:

```
8c11062d772b:/# cangw -L
cangw -A -s vcan2 -d vcan1 -X -e # 0 handled 0 dropped 832 deleted
cangw -A -s vcan1 -d vcan2 -X -e # 964 handled 0 dropped 0 deleted
```

normalno radi sve, sigurna opcija

12.3.

Golang podsjetnik

https://go.dev/tour/list
https://go.dev/doc/tutorial/call-module-code

- stdlib
  https://pkg.go.dev/std

Kako napisati docker network driver?

https://test-dockerrr.readthedocs.io/en/latest/extend/plugins_network/

primjer drivera:
https://github.com/tugbadm/docker-network-plugin/blob/master/mydriver.go

docs:
https://github.com/moby/moby/blob/master/libnetwork/docs/design.md
https://github.com/moby/moby/blob/master/libnetwork/docs/remote.md

Sandbox

A Sandbox contains the configuration of a container's network stack. This includes management of the container's interfaces, routing table and DNS settings. An implementation of a Sandbox could be a Linux Network Namespace, a FreeBSD Jail or other similar concept. A Sandbox may contain *many* endpoints from *multiple* networks.

Endpoint

An Endpoint joins a Sandbox to a Network. An implementation of an Endpoint could be a `veth` pair, an Open vSwitch internal port or similar. An Endpoint can belong to only one network and it can belong to only one Sandbox, if connected.

Network

A Network is a group of Endpoints that are able to communicate with each-other directly. An implementation of a Network could be a Linux bridge, a VLAN, etc. Networks consist of *many* endpoints.

CNM Lifecycle

Consumers of the CNM, like Docker, interact through the CNM Objects and its APIs to network the containers that they manage.

rivers register with NetworkController. Built-in drivers register inside of libnetwork, while remote drivers register with libnetwork via the Plugin mechanism (plugin-mechanism is WIP). Each driver handles a particular networkType.

NetworkController object is created using libnetwork.New() API to manage the allocation of Networks and optionally configure a Driver with driver specific Options.

Network is created using the controller's NewNetwork() API by providing a name and networkType. networkType parameter helps to choose a corresponding Driver and binds the created Network to that Driver. From this point, any operation on Network will be handled by that Driver.

controller.NewNetwork() API also takes in optional options parameter which carries Driver-specific options and Labels, which the Drivers can make use of for its purpose.

network.CreateEndpoint() can be called to create a new Endpoint in a given network. This API also accepts optional options parameter which drivers can make use of. These 'options' carry both well-known labels and driver-specific labels. Drivers will in turn be called with driver.CreateEndpoint and it can choose to reserve IPv4/IPv6 addresses when an Endpoint is created in a Network. The Driver will assign these addresses using InterfaceInfo interface defined in the driverapi. The IP/IPv6 are needed to complete the endpoint as service definition along with the ports the endpoint exposes since essentially a service endpoint is nothing but a network address and the port number that the application container is listening on.

endpoint.Join() can be used to attach a container to an Endpoint. The Join operation will create a Sandbox if it doesn't exist already for that container. The Drivers can make use of the Sandbox Key to identify multiple endpoints attached to a same container. This API also accepts optional options parameter which drivers can make use of.

Though it is not a direct design issue of LibNetwork, it is highly encouraged to have users like Docker to call the endpoint.Join() during Container's Start() lifecycle that is invoked before the container is made operational. As part of Docker integration, this will be taken care of.
One of a FAQ on endpoint join() API is that, why do we need an API to create an Endpoint and another to join the endpoint.
The answer is based on the fact that Endpoint represents a Service which may or may not be backed by a Container. When an Endpoint is created, it will have its resources reserved so that any container can get attached to the endpoint later and get a consistent networking behaviour.

- endpoint.Leave() can be invoked when a container is stopped. The Driver can cleanup the states that it allocated during the Join() call. LibNetwork will delete the Sandbox when the last referencing endpoint leaves the network. But LibNetwork keeps hold of the IP addresses as long as the endpoint is still present and will be reused when the container(or any container) joins again. This ensures that the container's resources are reused when they are Stopped and Started again.

  endpoint.Delete() is used to delete an endpoint from a network. This results in deleting an endpoint and cleaning up the cached sandbox.Info.

  network.Delete() is used to delete a network. LibNetwork will not allow the delete to proceed if there are any existing endpoints attached to the Network.
- pomocni docker network api go kod https://github.com/docker/go-plugins-helpers/blob/master/network/api.go
- docker go klijent
  - moglo bi biti korisno za pokretanje programa (cannelloni?) u containerima https://pkg.go.dev/github.com/docker/docker/client#pkg-overview

14.3 Pisanje drivera

napravljen repo za driver
https://github.com/lgrguricmileusnic/dockercan

dodan kao git podmodul u diplomski repo:
https://github.com/lgrguricmileusnic/diplomski/tree/master/rad/kod

Handshake

Handshake **rjesava Handler iz go helpera**

When loaded, a remote driver process receives an HTTP POST on the URL `/Plugin.Activate` with no payload. It must respond with a manifest of the form

```
{
	"Implements": ["NetworkDriver"]
}
```

Set capability

After Handshake, the remote driver will receive another POST message to the URL `/NetworkDriver.GetCapabilities` with no payload. The driver's response should have the form:

```
{
	"Scope":             "local"
	"ConnectivityScope": "global"
}
```

Value of "Scope" should be either "local" or "global" which indicates whether the resource allocations for this driver's network can be done only locally to the node or globally across the cluster of nodes. Any other value will fail driver's registration and return an error to the caller. Similarly, value of "ConnectivityScope" should be either "local" or "global" which indicates whether the driver's network can provide connectivity only locally to this node or globally across the cluster of nodes. If the value is missing, libnetwork will set it to the value of "Scope". should be either "local" or "global" which indicates

15.3.

Join

When a sandbox is given an endpoint, the remote process shall receive a POST to the URL `NetworkDriver.Join` of the form

```
{
        "NetworkID": string,
        "EndpointID": string,
        "SandboxKey": string,
        "Options": { ... }
}
```

The `NetworkID` and `EndpointID` have meanings as above. The `SandboxKey` identifies the sandbox. `Options` is an arbitrary map as supplied to the proxy.

The response must have the form

```
{
        "InterfaceName": {
                SrcName: string,
                DstPrefix: string
        },
        "Gateway": string,
        "GatewayIPv6": string,
        "StaticRoutes": [{
                "Destination": string,
                "RouteType": int,
                "NextHop": string,
        }, ...]
}
```

...

The entries in `InterfaceName` represent actual OS level interfaces that should be moved by LibNetwork into the sandbox; the `SrcName` is the name of the OS level interface that the remote process created, and the `DstPrefix` is a prefix for the name the OS level interface should have after it has been moved into the sandbox (LibNetwork will append an index to make sure the actual name does not collide with others).
...

If no gateway and no default static route is set by the driver in the Join response, LibNetwork will add an additional interface to the sandbox connecting to a default gateway network (a bridge network named *docker_gwbridge*) and program the default gateway into the sandbox accordingly, pointing to the interface address of the bridge *docker_gwbridge*.

- **obzirom da mi ne predajemo nikve rute niti gateway, stvoren ce biti i bridge izmedju containera**

Kreiranje namespacea (da bi vxcan sucelja ucinio privatnima):
https://medium.com/@tech_18484/how-to-create-network-namespace-in-linux-host-83ad56c4f46f