Anaconda network configuration:

### Stages:
1. **Networkdracut**: boot options, kickstart – ifcfg moved to Anaconda root
2. **Anaconda init**: Sectored rootfs
3. **Anaconda init**: kickstart type, activating kickstart config
4. **Anaconda post install tweaks and config** – ifcfg: ONBOOT, dracut boot options for network storage

#### Anaconda network configuration:
- Configuration in kickstart: device, hwaddr, tftp, dhcp
- Device specification (--device) and device binding (DEVICE/HWADDR/both)
- Special handling:
  - GUI can handle added devices (usb) to some extent (added_device callback), we don't support it officially
  - Initialization – sync with NM, waiting for connecting NM (dhcp) – not very robust but seems to work so far
  - GUI: nm-c-e <the con_uuid>, adding virtual device: nm-c-e team

#### Devices supported in installer (ethernet, infiniband, wifi, bond, team, bridge, vlan)

#### Bootloader options for root on network storage (iscsi)
- All ifcfgs copied to target system (ifcfg files, dhclient files, static routes configuration)
- One persistent configuration (connection ↔ ifcfg) per device. All the other connections should be only active connections of devices.
- Persistent configuration (ifcfg) of isntaller environment and target system is the same.
- Devices supported in installer (ethernet, infiniband, wifi, bond, team, bridge, vlan)

#### Anaconda post install tweaks and config – ifcfg: ONBOOT, dracut boot options for network storage
- GUI/TUI
- Anaconda initialize: kickstart %pre, activating kickstart config
- Anaconda initialize: settle started NM
- initramfs/dracut: boot options, kickstart → ifcfgs moved to Anaconda root

#### Network Manager starts
- NETWORK MANAGER STARTS
  - NetworkManager starting
  - Based on existing device configs and ifcfg files creates connections
  - Activates devices with autocons.

#### INTRAMFS/DRACUT
- **echo** network-device=ens7 static – activate
  - network-device=ens8 static
  - network-device=ens9 static
  - network-device=ens10 static
  - network-device=ens11 static
  - network-device=ens12 static
  - network-device=ens13 static
  - network-device=ens14 static
  - network-device=ens15 static

#### intramfs/dracut
- **network** –device=ens7 static – activate
- **network** –device=ens8 static
- **network** –device=ens9 static
- **network** –device=ens10 static
- **network** –device=ens11 static
- **network** –device=ens12 static
- **network** –device=ens13 static
- **network** –device=ens14 static
- **network** –device=ens15 static

#### Network Manager starting
- Based on existing device configs and ifcfg files creates connections
- Activates devices with autocons.

#### Network Manager starting
- NetworkManager starting
- Based on existing device configs and ifcfg files creates connections
- Activates devices with autocons.
NETWORK MANAGER STARTS

ANACONDA - Network Initialize

* This all operates only with devices supported by anaconda.
* In most cases ifcfgs are manipulated via NM connections, sometimes directly (setOnboot).

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.

We use call this in %TUI to be sure – added USB devices, timing issues...

For devices without ifcfgs it creates ones. It dumps its default in-memory autoconnection (Wired connection 1) and changes its configuration to handle correct settings of ONBOOT policy at the end of installation, for GUI (mapping devices to their single configuration/connection). We even call this in %TUI to be sure – added USB devices, timing issues...

GOTCHA: when creating the connection we have to set ONSBOOTH (autoconnect=false) so that the device is not activated when it is added if not required. And handle correct settings of ONSBOOTH later. This is similar to (mis)using ONBOOT for –activate in Kickstart.
ANACONDA – GUI

● We display only supported devices (and no slaves) in a list of DeviceConfiguration objects.

● The object maps a device (libnm device object) to its ifcfg connection (libnm connection object) which is the single connection that can be edited with [Configure] (→ nm-c-e connection_uuid).

● Creating the list (DeviceConfiguration objects) and updating it during changes (adding virtual devices) is tricky. For wired devices (eth, infiniband) the device is primary in DeviceConfiguration. For virtual devices (bond, vlan, team, bridge) it is the connection which is primary/persistent (and can be added/removed with [+][-]). Virtual devices are ephemeral.

● UI shows current device configuration which can be different from persistent ifcfg connection (ens14, ens15, ens9). It can even have its own different (in-memory) connection (ens14 from NM start or ens9 from dracut) active. The in-memory connection will be replaced by the persistent ifcfg one (but not deleted). Compare with ens7 and ens9: the device is recreated by NM (ifens5 and ifens9) OR with a config file change. Alternatively, the in-memory connection is automatically recreated with edited connection (users requirement).

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.

● After editing the device is automatically reactivated with edited connection (users requirement)!

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.

● On apply we generate ksdata.network for each DeviceConfiguration in the list. We do it by parsing ifcfg files, not from NM connections. Why? mainly because of ONBOOT? Not sure. Also we set –activate based on whether the device is active now.
Ifcfg-ens7
Uuid1
static (ks)
ONBOOT=yes
(activate)
DEVICE=ens7

Ifcfg-ens9
Uuid2
static (ks)
ONBOOT=no
(not activate)
DEVICE=ens9

Ifcfg-ens10
Uuid3
dhcp (ks)
ONBOOT=yes
(activate)
DEVICE=ens11

Ifcfg-ens11
Uuid4
static (ks)
ONBOOT=yes
(activate)
ONBOOT=no
DEVICE=ens11

Ifcfg-ens12
Uuid10
static (ks)
ONBOOT=no
(not activate)
DEVICE=ens12

Ifcfg-ens14
Uuid11
static (ks pre)
ONBOOT=no
(not activate)
ONBOOT=yes
DEVICE=ens14

Ifcfg-ens15
Uuid15
static (ks pre)
ONBOOT=yes
DEVICE=ens15

Ifcfg-ens13
Uuid9
dhcp (NM default)
ONBOOT=yes
DEVICE=ens13

Ifcfg-ens3
Uuid0
Dhcp (bopt)
ONBOOT=yes
DEVICE=ens3

---

**Network information**

network --bootproto=dhcp --device=ens10 --ipv6=auto --activate
network --bootproto=static --device=ens12 --gateway=192.168.124.255 --ip=192.168.124.203 --nameserver=10.34.39.2 --netmask=255.255.255.0 --ipv6=auto
network --bootproto=dhcp --device=ens13 --ipv6=auto --activate
network --bootproto=static --device=ens14 --gateway=192.168.124.255 --ip=192.168.124.204 --nameserver=10.34.39.2 --netmask=255.255.255.0 --onboot=off --ipv6=auto --activate
network --bootproto=static --device=ens15 --gateway=192.168.124.255 --ip=192.168.124.205 --nameserver=10.34.39.2 --netmask=255.255.255.0 --ipv6=auto
network --hostname=localhost.localdomain

---

**Ksdata.network**

ens7
static
(NM)
ens9
dhcp
dracut
ens10
dhcp
dracut
ens11
static
(NM)
ens12
static
(GUI)
ens13
dhcp
NM default
ens14
static
(GUI)
ens15
dhcp
dracut
ens3
dhcp
dracut
ens15
dhcp
dracut

---

**InConda – network.execute → write_nework_config**

2) Instclass.setNetworkOnbootDefault
    ● Fedora – ensure at least one (first device with link) device has ONBOOT=yes (eg for media installs)
    ● RHEL – ensure at least one (the one with default route) device has ONBOOT=yes (eg for media installs) – it used to differ in some other aspect from Fedora IIRC
    Doing this in Anaconda root could activate the devices by change to ONBOOT=yes.
    No change in this example.

---

**Dracut boot options for root on net (iscsi):**

- for iSCSI that needs to be connected in initramfs (root on iSCSI) we need to supply dracut ip= boot options

Dracut boot options for root on net (iscsi):

- ip=ibft

---

**Anaconda → network.execute → write_nework_config**

- For iSCSI it is done in storage
- For iSCSI that needs to be connected in initramfs (root on iSCSI) we need to supply dracut ip= boot options

Dracut boot options for root on net (iscsi):

- ip=ibft

---

**Network information**

network --bootproto=dhcp --device=ens10 --ipv6=auto --activate
network --bootproto=static --device=ens12 --gateway=192.168.124.255 --ip=192.168.124.203 --nameserver=10.34.39.2 --netmask=255.255.255.0 --ipv6=auto
network --bootproto=dhcp --device=ens13 --ipv6=auto --activate
network --bootproto=static --device=ens14 --gateway=192.168.124.255 --ip=192.168.124.204 --nameserver=10.34.39.2 --netmask=255.255.255.0 --onboot=off --ipv6=auto --activate
network --bootproto=static --device=ens15 --gateway=192.168.124.255 --ip=192.168.124.205 --nameserver=10.34.39.2 --netmask=255.255.255.0 --ipv6=auto
network --hostname=localhost.localdomain