RtBrick Tools Installation Manual - version 20.6.1-rc0

Version	Date	Description
1.0	16.03.2020	Initial version
1.1	01.04.2020	The following changes were made: • Updated `configuration.yaml` section, as a result of the release of rtb-ansible 3.0.0. • Added the *Connecting a container to outside world* section, with examples
1.2	25.05.2020	Updated the document with the latest version of the image, which is `20.5.1-rc0`
1.3	15.06.2020	Updated the document with the latest version of the image, which is `20.6.1-rc0`

Introduction

To speed up the process of container bring up, we are providing an ansible based solution to spawn and configure your containers and the apps in them. This is done using two packages RtBrick offers, `rtbrick-ansible` and `rtbrick-image`. Using them, the time to bring up RtBrick container routers is shortened, because there is no need to manually setup and create configuration files, or to set up containers manually.

Installation

The installation of RtBrick utilities is broken into several steps, as follows:

The following commands and outputs are validated for Ubuntu 18.04 LTS release.

Step 1: Removing any existing RtBrick packages

The rtb-* utilities validated with Ubuntu 18.04 are the second (third, for `rtb-ansible`) iteration, so you may already have older versions installed. The differences between versions are big enough that there is no backwards compatibility, so this step comes in:

```
apt list --installed | egrep -i rtbrick | awk -F '/' '{print $1;}' | xargs sudo apt remove -y
```

Among other output, you will get the following:

```
The following packages will be REMOVED:
  rtbrick-ansible rtbrick-imgstore rtbrick-lxc-tools
```

Step 2: Add new RtBrick GPG signing key

Notice the URL called in the command: it should always be https://releases.rtbrick.com/

```
curl -fsSL https://releases.rtbrick.com/security/RtBrick-Support.pubkey.asc | sudo apt-key add
```

-

Step 3: Adding the correct RtBrick repositories

Remember that in Step 1 we removed the old packages; the same must be done for any existing rtbrick APT repo URLs (as well as adding the new ones):

```
echo 'deb [ arch=amd64 ] http://releases.rtbrick.com/_/20.6.1/ubuntu/rtbrick-tools bionic rtbrick-tools' | sudo tee /etc/apt/sources.list.d/rtbrick.list
```

We then have to update the package list: `sudo apt update`

Step 4: Install `rtbrick-ansible`

One very important dependency of *rtbrick-ansible* is Ansible itself. For installing Ansible, you can use the official documentation, which can be found at https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-ansible-on-ubuntu. **Make sure you have the latest version of Ansible installed, before installing `rtb-ansible`!**

```
sudo apt install -y rtbrick-ansible
```

The *rtbrick-ansible* package contains the `rtb-ansible` command, which is used for actual deployment of the containers.

Currently the only option that is properly maintained in the `rtb-ansible` command is `full-setup`. Support for the other options will come in future releases.

It is important to note that, with version 2.0, *rtbrick-ansible* has as dependency (and installs) a new RtBrick package, called *rtbrick-imgstore*, which we'll discuss next.

Rtbrick-imgstore package

This package is RtBrick's image store handling tool. An image store (imgstore) is a versioned, checksumed and cryptographically signed store of versioned files. It was developed and optimized with the primary goal of storing and distributing Linux OS and Linux container images however it can be used to store any kind of files.

An image store is for images what an APT repository is for debian packages. It also has some similarities with a docker registry (not to be confused with a docker repository).

This tool provides the `rtb-image` command, used for interacting with an image store accesible via HTTP, making a local cache of that image store, which can later be used to start LXC containers.

```
pinky@tattooine:~$ sudo apt search rtbrick-imgstore
Sorting... Done
Full Text Search... Done
rtbrick-imgstore/bionic,now 0.4.1 amd64 [installed,automatic]
  RtBrick image store handling tool

pinky@tattooine:~$ sudo apt show rtbrick-imgstore
Package: rtbrick-imgstore
Version: 0.4.1
Priority: extra
Section: rtbrick-internal
Maintainer: RtBrick Support <support@rtbrick.com>
Installed-Size: 24.1 MB
Provides: rtbrick-imgstore
Depends: liblxc-common, liblxc1, lxc, zstd
Replaces: rtbrick-imgstore
Download-Size: 8786 kB
APT-Manual-Installed: no
APT-Sources: http://releases.rtbrick.com/_/20.6.1-rc0/ubuntu/rtbrick-tools
bionic/rtbrick-tools amd64 Packages
Description: RtBrick image store handling tool
 rtbrick_package_properties:
 version: 0.4.1
 branch: master
 commit: 1b14aa3e49b5b35a41899e20f73340b9d34b780d
 commit_timestamp: 1584356254
 commit_date: 2020-03-16 10:57:34 UTC
 build_timestamp: 1584356367
 build_date: 2020-03-16 10:59:27 UTC
 build_job_hash: 423be4f25ec9
 git_dependencies:
 - git_dep: gopackages/imgstore @ master > imgstore
 git_dep_branch: master
 git_dep_commit: 7f0eac0104646c4d067d3849513d4f75364455a8
```

The tool (the binary) has in it embedded the GPG public key of support@rtbrick.com , identity which is used to sign all RtBrick images and the image store itself.

Common usage of `rtb-image`

`rtb-image` has enough versatility, but a few options are commonly used:

- `containers list` - List all the LXC containers which are created on the **local** system.
- `show [<flags>] <UUID>` - Show details of image identified by UUID. By default this shows the image in the local cached copy of the store.
- `run --name=NAME [<flags>] <UUID>` - Run an LXC container using the specified image. The container must not be already created.

- `list [<flags>] <UUID>` - List all the images in the store. By default this lists in the images in the local cached copy of the store.

Table 1. rtb-image list flags

Value	Description
-o, --remote	List images directly from the remote store and not from the local cached copy.
-d, --detailed	List detailed information about images.
-f, --format=FORMAT	List only images with a specific format.
-r, --role=ROLE	List only images with a specific role. Currently, roles are spine and leaf.
-p, --platform=PLATFORM	List only images for a specific platform.
-v, --ver-range=VER-RANGE	List only images versions that fall in the provided version range. See the syntax for version ranges at https://godoc.org/github.com/blang/semver#Range . The hardcoded strings 'latest' or 'newest' will always filter down to a single image, the one considered the newest according to the sorting rules for versions.
-l, --limit=LIMIT	Limit the list of returned images to the the l newest images.

An important part of `rtb-image` is that it is used to create a local cache of the remote RtBrick image repo. This is done using the `rtb-image update` command:

```
sudo rtb-image update
2020/03/16 13:49:54 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-
rc0/index.sha512
2020/03/16 13:49:54 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-
rc0/index.asc
2020/03/16 13:49:54 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-rc0/index
Local cached copy updated to: Store: /var/cache/rtbrick/imagestore Version: 0.1.4
ValidUntil: 2020-05-17 13:25:24.443775551 +0000 UTC
```

Then we can list the local copies:

```
pinky@tattooine:~$ rtb-image list

Store: /var/cache/rtbrick/imagestore Version: 0.1.4 ValidUntil: 2020-05-17
13:25:24.443775551 +0000 UTC

UUID                                   Version     Filename
Format  Role        Platform    Cached
4838fd65-c4b6-4d05-a372-ac0334f3623b    20.6.1-rc0-rc0  rbfs-cont/rbfs-spine-virtual-
20.6.1-rc0-rc0.tar.zst         lxd     spine       virtual     false
0e2194a9-4cbd-484b-a1a5-4b2c13dc1ccf    20.6.1-rc0-rc0  rbfs-cont/rbfs-accessleaf-
virtual-20.6.1-rc0-rc0.tar.zst    lxd     accessleaf  virtual     false
638a28bb-7ee8-460f-8fe6-9ec8d4337894    20.6.1-rc0-rc0  rbfs-cont/rbfs-spine-qmx-
20.6.1-rc0-rc0.tar.zst             lxd     spine       qmx         false
21ce3b5c-1e18-474a-8456-06e431da158d    20.6.1-rc0-rc0  rbfs-cont/rbfs-accessleaf-qmx-
20.6.1-rc0-rc0.tar.zst        lxd     accessleaf  qmx         false
```

Image formats and ONL image installation for supported hardware

RtBrick images delivered through the RtBrick image store and the `rtb-image` utility have 3 main attributes:

- `format`: This is the file format of in which the image is packaged and archived.
- `role`: The role inside a network of the device which will be running the image.
- `platform`: Identifies the hardware platform or virtualized environment in which the image can run.

RtBrick images mean to be used as containers in a virtualized environment will have `format` == `lxd` and `platform` == `virtual`.

RtBrick images mean to be installed on supported whitebox switch hardware devices will have `format` == `onl-installer` and `platform` set accordingly to the specific switching hardware.

You can see this using `rtb-image list` command and looking for the `Format` column.

ONL images

ONL images are generally installed using a Zero Touch Provisioning (ZTP) server. The Installation section applies for both virtual and harware installations, with the difference that, when having a physical deployment (One with a ZTP server and switched running ONL images) we can install just the `rtbrick-imgstore` package on the ZTP server, since it doesn't have Ansible as dependency (Ansible not being a part of the default Ubuntu repositories), and because generally you will not have containers running on the ZTP server itself.

A typical ONL image download will look as in the following snippet:

```
pinky@tattooine$ sudo rtb-image update
2020/03/17 07:06:41 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-
rc0/index.sha512
2020/03/17 07:06:42 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-
rc0/index.asc
2020/03/17 07:06:42 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-rc0/index
Local cached copy already up to date: Store: /var/cache/rtbrick/imagestore Version:
0.1.10 ValidUntil: 2020-05-17 18:27:28.624270218 +0000 UTC

$ rtb-image list --format onl-installer --platform qmx --role spine --ver-range latest

Store: /var/cache/rtbrick/imagestore Version: 0.1.10 ValidUntil: 2020-05-17
18:27:28.624270218 +0000 UTC

UUID                                 Version     Filename
Format       Role Platform Cached
c23c4095-5b16-4535-9786-16436a0273d3 20.6.1-rc0-rc0.1 rtbrick-onl-installer/rtbrick-
onl-installer-spine-qmx-20... onl-installer spine qmx     false

pinky@tattooine$ sudo rtb-image pull c23c4095-5b16-4535-9786-16436a0273d3
2020/03/17 07:07:09 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-
rc0/index.sha512
2020/03/17 07:07:09 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-
rc0/index.asc
2020/03/17 07:07:09 [DEBUG] GET http://releases.rtbrick.com/_/images/20.6.1-rc0/index
rtbrick-onl-installer-spine-qmx-20.6.1-rc0-rc0.1.sha512 207 B / 207 B
[=============================================================================
=====] 100.00% 0s
rtbrick-onl-installer-spine-qmx-20.6.1-rc0-rc0.1.asc 833 B / 833 B
[=============================================================================
========] 100.00% 0s
rtbrick-onl-installer-spine-qmx-20.6.1-rc0-rc0.1 1.53 GiB / 1.53 GiB
[=============================================================================
======] 100.00% 23s
rtbrick-onl-installer-spine-qmx-20.6.1-rc0-rc0.1: decompressing 100 B / 100 B
[=============================================================================]
100.00% 0s

pinky@tattooine$ rtb-image show c23c4095-5b16-4535-9786-16436a0273d3

Store: /var/cache/rtbrick/imagestore Version: 0.1.10 ValidUntil: 2020-05-17
18:27:28.624270218 +0000 UTC

UUID:         c23c4095-5b16-4535-9786-16436a0273d3
Version:       20.6.1-rc0-rc0.1
Filename:     rtbrick-onl-installer/rtbrick-onl-installer-spine-qmx-20.6.1-rc0-rc0.1
FullPath/URL: /var/cache/rtbrick/imagestore/rtbrick-onl-installer/rtbrick-onl-
installer-spine-qmx-20.6.1-rc0-rc0.1
SHA512:
d4d7dfa52bfb644914a4e83d40683503cd77076df44316eeee5ed23fe7d72840abff716909ca8d29b9fbc7
```

```
dc8defcd95d50d60fd075352a945a56e14dc25d91a
Format:       onl-installer
Role:         spine
Platform:     qmx
Cached:       true
ExtractedPath:
```

In a design where the download of the image happens on a different server than the ZTP used for the actual installation, we can install the *rtbrick-imgstore* package, and move by some means (`rsync`, for example) the images from `var/cache/rtbrick/imagestore/` of that internet-connected to the ZTP server.

Configuration.yaml file

configuration.yaml is one of the necessary files used to sucessfully spin up RtBrick containers. It resides in the root of the topology folder, and specifies the options based on which the containers are built. The structure of the file will be discussed in the following sections. It is worth nothing that, starting with version 2.0 of the `rtbrick-ansible` package, the **configuration.yaml** has also been upgraded. You can verify the version of rtbrick-ansible that you are using with `apt list --installed | grep rtbrick-ansible`.

The new format is not compatible with older releases. The syntax is different, and old options are not supported, i.e. *silently ignored*. One reason for this is that the new RBFS container images come preinstalled with all RtBrick packages that are needed for that type of image (more on RtBrick image types later).

Structure of configuration.yaml

Each configuration.yaml file has the following general structure:

```
containers:
    <name_of_the_machine_on_which_the_container_runs>:
        <container_name>:
            role: [spine|leaf]
            platform: [virtual|qmx]
            version:
            extra_veth:
            lxc_config:
host_config:
    <name_of_the_machine_on_which_the_container_runs>:
        setup: [true|false]
        type: "default"
```

As shown above, it has 2 configuration parts: the container part and the host configuration part. We'll discuss each separately.

The container section

A word on *role* and *platform*

The *role* and *platform* options are RtBrick container options that deal with the overall network topology (read: where in the network you place the container).

These two options are mandatory! Without them, the `rtb-ansible` command will exit with an error and the containers will not be created.

The *role* option has 2 possible values, which (of course) are mutually exclusive:

Table 2. Role parameter

Value	Description
spine	Role used for spine containers
leaf	Role used for leaf containers

The *platform* is used to tell rtb-ansible the "host" on which the containers are spun on; this is important because, based on the *platform* option rtb-ansible knows to pull the correct image (each imagine being compiled for its respective platform, it will not work on another, e.g. a "virtual" image will not work on a switch). It currently has 2 possible values:

Table 3. Platform parameter

Value	Description
virtual	Used when the containers are spun on a server/laptop/computer. Mainly used for testing/demonstration purposes.
qmx	Used when containers will start on an actual switching platform, with Qumran-MX (QMX) chipset.

Other container parameters

Besides *role* and *platform*, containers can have other parameters, which are not mandatory. The following are currently available for configuration:

Table 4. Other containers parameters

Value	Description
version	Version is used to control the image version that a container runs. If not supplied, it will default to *latest*, as defined by rtb-image. In a configuration.yaml file, multiple containers can have different running RtBrick images

Value	Description
extra_veth	This is a list of veth configuration dicts used to add more veths into the container. *extra_veth* is useful when wanting to connect the container with something not-RBFS specific (for example with a container running a RADIUS server). It accepts the following options: • `cont_intf`: Name of the veth interface inside the container. If not provided it defaults to eth<num> model. • `host_intf`: Corresponding interface name on the host (outside the container) (cont_intf <→ host_intf). If not provided it defaults to a veth<random letters> model. • `bridge_intf`: This provides the bridge interface to which the container (via host_intf) is connected (thus this needs to be the same for each 2 containers that you want to connect). • `physical_intf`: *Optional*, a physical interface of the host to the added to the bridge • `ipv4_address`: IPv4 address of the *veth* interface
lxc_config	A set of LXC configs to be processed inside the container. Currently it accepts only one option: • `append`: list of string to be appended to the config

Connecting a container to outside world with `extra_veth`

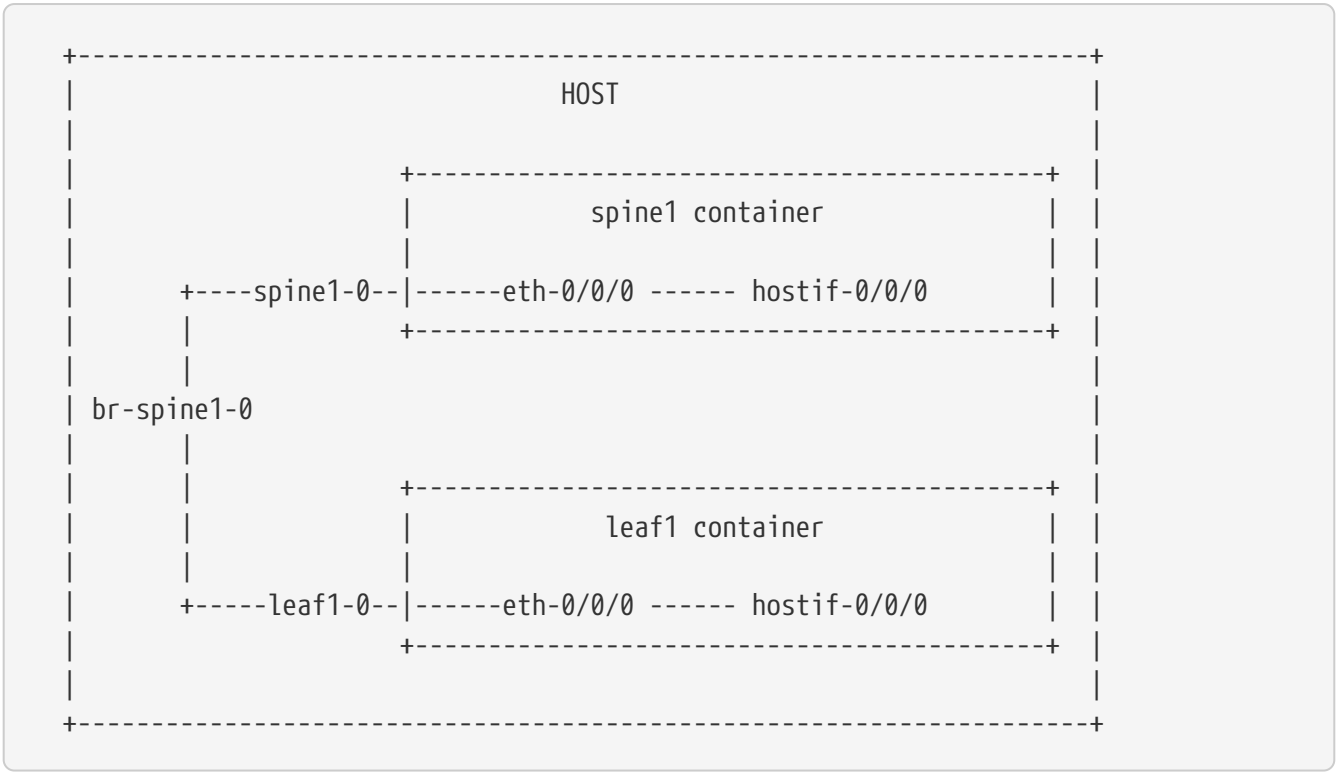
A major upgrade introduced starting `rtb-ansible` 3.0.0, is the capability to connect an RtBrick container to a live network, *without the need for DPDK*, via `physical_intf`, which is a host physical network interface that will be added to the bridge. Of course, the current use case of `extra_veth`, which is to connect 2 running containers will continue to work. For such a use case **`physical_intf` must not be present** in the configuration. For this, `bridge_intf` needs to have the same value for the 2 containers that need to be connected. WARNING: With the current way bridge interfaces are set up by `rtb-ansible`, bridges are VLAN transparent. If you need multiple VLANs there is no need to create multiple bridges (assuming that all VLANs need to be available on the same RBFS physical interface). This works for both for the container-to-container use case and the container-to-physical one. The following sections cover some examples about how `extra_veth` option is used

Example 1 - two containers running on the same host

Lets say we want to connect containers named *spine1* and *leaf1*; we will have the following interfaces:

- `hostif-0/0/0` - the interface as seen in RBFS (in other words, by VPP)
- `eth-0/0/0` - `cont_intf` from the `configuration.yaml` file (`extra_veth` option)
- `spine1-0` - `host_intf` from the `configuration.yaml` file (`extra_veth` option)
- `br-spine1-0` - `bridge_intf` from the `configuration.yaml` file (`extra_veth` option)

`hostif-0/0/0` and `host_intf` **have similar** names, be sure not to confuse them!



So, for the above setup, the `configuration.yaml` would look like this:

```
        spine1:
            role: spine
            platform: virtual
            version: "latest ~=g8daily.*Bmaster"
            extra_veth:
                - cont_intf: "eth-0-0-0"
                  host_intf: "spine1-0"
                  bridge_intf: "br-spine1-0"
        leaf1:
            role: leaf
            platform: virtual
            version: "latest ~=g8daily.*Bmaster"
            extra_veth:
                - cont_intf: "eth-0-0-0"
                  host_intf: "leaf1-0"
                  bridge_intf: "br-spine1-0"
```

notice the lack of `physical_intf` parameter

Example 2 - one container connecting to the outside world

For example, we have host physical network interfaces `ens37` and `ens39` which we want to use inside container *spine1* as interfaces `hostif-0/0/0` and `hostif-0/0/1`. The initial setup would look like this:

```
  +-------------------------------------------+
  |                  HOST                     |
  |                                           |
  |                                           |
  |                    +------------------+   |
  |                    |      spine1      |   |
  |                    |     container    |   |
  |                    |                  |   |
  |                    |                  |   |
ens37                  | hostif-0/0/0     |   |
  |                    |                  |   |
  |                    |                  |   |
  |                    |                  |   |
  |                    |                  |   |
ens39                  | hostif-0/0/1     |   |
  |                    |                  |   |
  |                    +------------------+   |
  |                                           |
  |                                           |
  +-------------------------------------------+
```

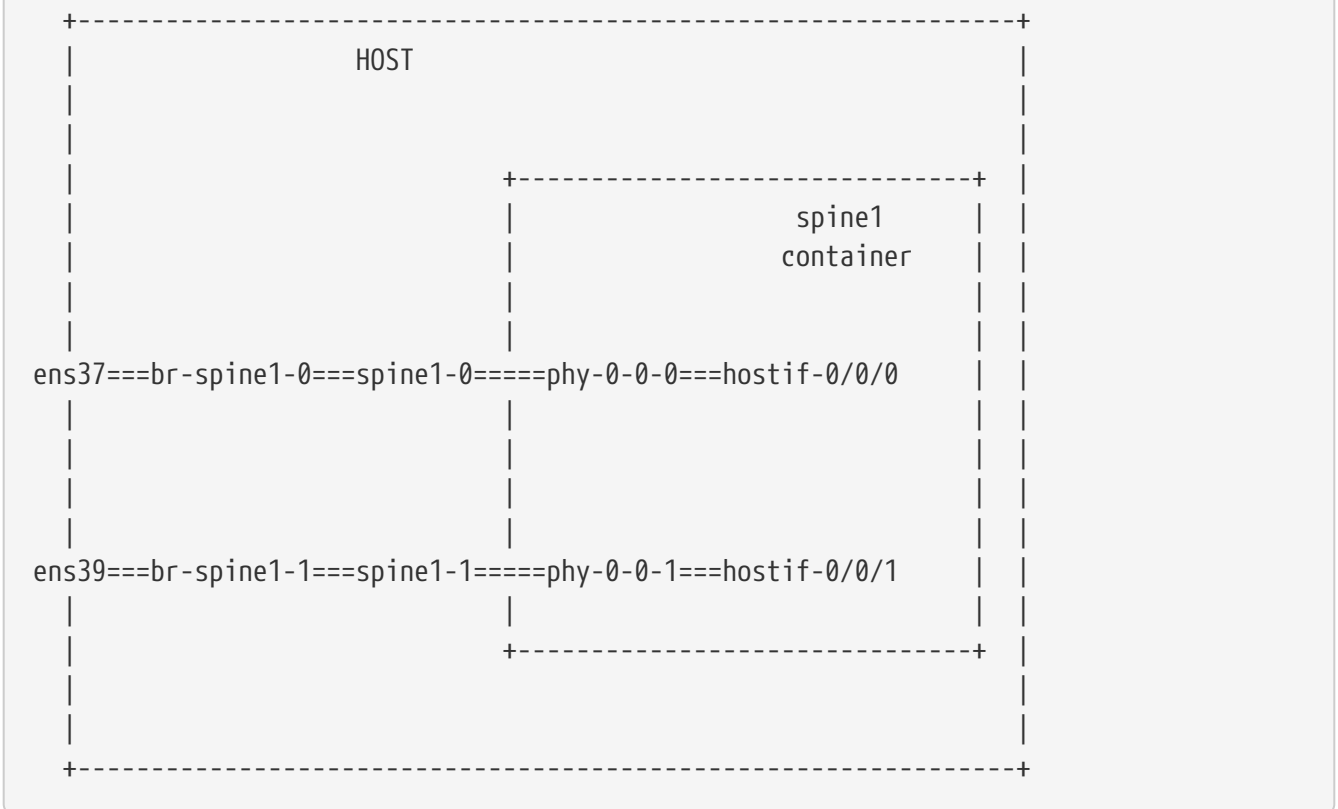
In `configuration.yaml` this assigns host physical network interfaces `ens37` and `ens39` to be used inside the *spine1* container:

```
        spine1:
            role: spine
            platform: virtual
            version: "latest ~=g8daily.*Bmaster"
            extra_veth:
                - cont_intf: "phy-0-0-0"
                  host_intf: "spine1-0"
                  bridge_intf: "br-spine1-0"
                  physical_intf: "ens37"
                - cont_intf: "phy-0-0-1"
                  host_intf: "spine1-1"
                  bridge_intf: "br-spine1-1"
                  physical_intf: "ens39"
```

In `configs/localhost-spine1.yaml` this creates the corresponding RBFS interfaces `hostif-0/0/0` and `hostif-0/0/1`:

```
pre_program_steps:
  - type: "Shell Command"
    command: "rtb confd set interface physical hostif-0/0/0 host-if phy-0-0-0"
  - type: "Shell Command"
    command: "rtb confd set interface physical hostif-0/0/1 host-if phy-0-0-1"
```

The final topology will be:



`hostif-0/0/0` and `hostif-0/0/1` can be used as any other RBFS interface:

```
 - ifp_name: "hostif-0/0/1"
                  interface_description: "Physical interface connecting between leaf1
and spine1"
                  logical:
                       - logical_unit_id: 23
                         ifl_name: "hostif-0/0/0/23"
                         instance: "default"
                         outer_vlan_id: 230
                         interface_description: "Connecting towards Leaf 1"
                         family:
                             ipv4:
                                 address:
                                     - prefix4: "23.1.1.2/24"
                             ipv6:
                                 address:
                                     - prefix6: "23:1:1::2/64"
```

The *host_config* section

This section onf the *configuration.yaml* file contains the host specific configurations. The available options for this section are listed in the following snippet:

```
host_config:
    <name_of_the_machine_on_which_the_container_runs>:
        setup: [true|false]
        type: "[default|dpdk|onl-bcm]"
        clearshm: [true|false]
        host_ip: "IPv4 address"
```

Of *host_config* section, the `setup` and `type` options are **mandatory**!

Each option is explained in the table below:

Table 5. Host configuration options

Value	Description
setup	This is a boolean value. If set to true, this will initalise the host based on the type provided. This can be set to be a one time initialization and then reset to false once done.

Value	Description
type	This option specifies the host type supported, hosts being set up based on the value present here. Three types are currently supported: • `default`: This is the vanilla VPP software based system. Only VPP memif interfaces are supported for containers in a host of this type. • `dpdk`: This is the VPP DPDK based system. The containers will additionally support dpdk enabled interfaces using the device pci id list provided in the container configuration yaml file in the configs folder. • `onl-bcm`: This is the ONL based system supporting broadcom chipsets.
clear_shm	This is a boolean value. If set to true, it clears the `/shm` folder everytime rtbrick-setup.yaml or rtbrick-setup-containers.yaml is called.
host_ip	This is an string value containing an IP address. This is to specify an IP address used by ansible as an inventory ssh host access. This is primarily used in Cloud instances where the ansible ip is usually a private IP and you need to program using the public floating ip linked to the instance.

A working example of `host_config` section can be as simple as

```
host_config:
    localhost:
      setup: false
      type: "default"
```

Servers file

The `servers` file is `rtb-ansible` inventory file. It contains the list of containers that are spun up during `rtb-ansible` run. There are three group names:

- `[config_host]` - this usually contains localhost configurations. You can manually add extra options for localhost, or you can add a remote host.
- `[container_set]` - this contains information automatically added by `rtb-ansible` script. This part is constructed with the information provided in the `configuration.yaml` file. This section **MUST**

NOT be edited manually!

- `[container_frrs]` - legacy group name, used for FRR containers. Not supported anymore.

In all three groups different variables can exist; they are the ones that are native to Ansible, and fall in one of the following categories:

- for a user authentication (the ones, but not limited to, below):
  - `ansible_ssh_user`
  - `ansible_ssh_pass`
  - `ansible_ssh_extra_args`
  - `ansible_ssh_extra_args`
- for host definition
  - `ansible_host`
- remote host environment parameters
  - `ansible_python_interpreter`

please note that the variables are the ones used previous of Ansible 2.0

Besides these, you may also find RtBrick-defined variables, like `current_container_name`, `current_host_type`, or `container_frr`. These are used in the internal workings of `rtb-ansible`, and if present, their values should not be modified without RtBrick engineering support.

`[config_host]` section

A "classic" config for this section would look like this:

```
# ================================================
[config_host]
localhost ansible_connection=local ansible_sudo_pass=ubuntu
```

As seen in the snippet above, it tells `rtb-ansible` what `sudo` password should use, in cases in which, e.g., the sudo password is not yet cached. We can also have a configuration line for a remote machine; this is a common example:

```
[config_host]
leaf1 ansible_ssh_host=192.168.1.246 ansible_ssh_user=ubuntu
ansible_ssh_pass=/home/ubuntu/.ssh/id_rsa ansible_ssh_port=22 ansible_sudo_pass=leaf1
```

`[container_set]` section

Created by `rtb-ansible`, this section should not be manually edited by the user. A line will be created for every container in the topology. A typical line in this group will be something like this:

```
localhost-alba ansible_ssh_host=10.0.3.183 ansible_ssh_user=ubuntu
ansible_ssh_pass=ubuntu ansible_ssh_extra_args= current_host_name=localhost
current_container_name=alba current_host_type=default current_container_type=
container_frr=False
```