acontis technologies GmbH

SOFTWARE

RT-Linux

acontis Real-time Linux guest

Version 9.x

Edition: July 29, 2025

Table of Contents

1 Introduction

This guide describes how to set up and work with a `Real-time Linux` guest in the *acontis Hypervisor*.

2 RT-Linux Guest

2.1 Example Real-time Linux guest

Initially, the Hypervisor Host does not provide any example guest folders. To switch to an example guest, you must execute the corresponding initialization. For instructions on how to initialize the examples, refer to Hypervisor Manual (chapter *Example guest folders*).

Important: You need at least **256 Mbyte** RAM for the shipped RT-Linux image.

After the example is initialized, the Real-time Linux guest can be found in `/hv/guests/guestrtlinux`

```
hv_open_example rt-linux
hv_sync_example rt-linux
cd /hv/guests/guestrtlinux
```

Real-time Linux guest specific files are stored in `/hv/guests/guestrtlinux/files`:

- `autostart.sh`: RT-Linux autostart script, you may enhance this script to automatically run your RT-Linux application.
- `timesync.sh`: Time-synchronization script, this will assure the RT-Linux time and date is consistent with the Hypervisor Host.
- `remote_cmd_exec.sh`: Example script which is used to show how to run a RT-Linux application from within a Windows or Ubuntu guest.

2.1.1 File system access for RT-Linux guests

The Hypervisor Host file system is mounted into the `/mnt/rtfiles` folder of RT-Linux. The **Hypervisor Host** folder `/hv/guests` acts as the root folder for all guests and is mounted into this folder.

2.2 Example Real-time Linux guest with Docker

To use Real-time Linux guest with Docker you must initialize the appropriate Real-time Linux guest first. Refer to *Example Real-time Linux guest*.

Open the System Manager in a browser and verify that System Manager is connected, synchronized and is in Config mode. In the Navigator tab open the example Real-time Linux guest. Change Hardware/Memory to at least 1200 Mb and change the RTOS Image to `Linux 5.15 x64 Docker`.

Important: You need at least **1200 Mbyte** RAM for the shipped RT-Linux image.

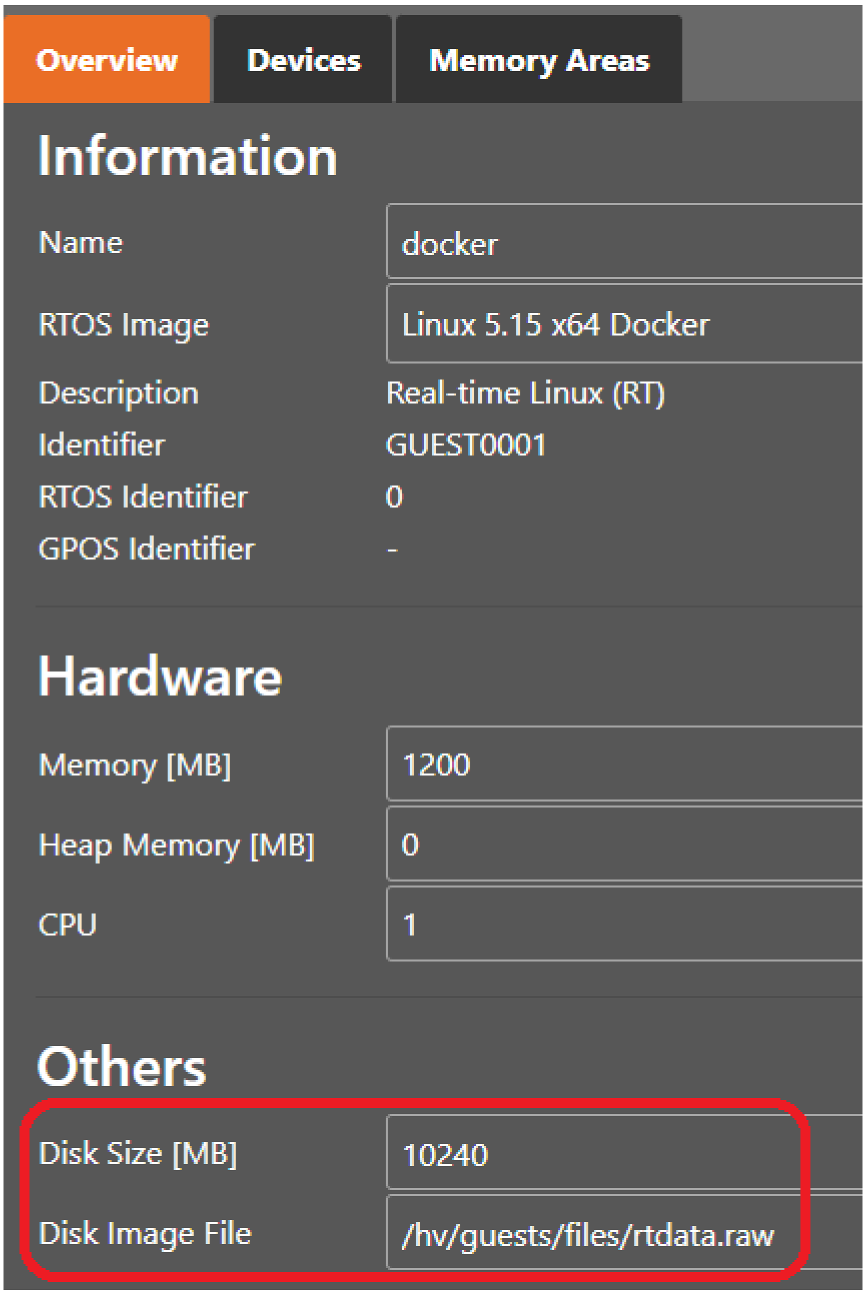
Hint: The RT-Linux Docker repository can be created either in RAM or in a persistent filesystem. If Docker is used in RAM, the repository must be set up fresh after each restart of RT-Linux. To make the

Docker repository persistent, it must be stored in a persistent filesystem container. The desired size of the container can be determined using the System Manager or, alternatively, be set in `usr_guest_config.sh` using the `rt_data_size_MB` switch. When RT-Linux starts, the filesystem is mounted at `/mnt/rtdata`, and the Docker repository is mounted into this filesystem. The default location of the container file is below the guest folder, e.g. `/hv/guests/guest0001/files/rtdataimage.raw`.

Important: The data file will only be created when the RT-Linux guest is started and no existing data container file is found. Once the container file is created, its size cannot be changed. The size should be large enough for your complete Docker environment. If the default path and filename for the container file shall be changed, you need to assure that it must be located below the `/hv/guest` folder!

Configure the size of the data file. In this guide, we will create a data file with a size of 10GB (10240MB).

You can configure the data container path, filename and size using the System Manager:



Information

Name	docker
RTOS Image	Linux 5.15 x64 Docker
Description	Real-time Linux (RT)
Identifier	GUEST0001
RTOS Identifier	0
GPOS Identifier	-

Hardware

Memory [MB]	1200
Heap Memory [MB]	0
CPU	1

Others

Disk Size [MB]	10240
Disk Image File	/hv/guests/files/rtdata.raw

Or, alternatively, using the *usr_guest_config.sh* file:

```
$ cd /hv/guests/guest0001
$ mousepad usr_guest_config.sh
```

Add or Adjust the following lines (in this case, the default path and filename will be used):

```
export rt_data_size_MB=10240
```

Turn the System Manager into Run mode, start the VMF and start the guest. Connect to the guest using ssh or telnet. Next, verify that Docker works by running `docker info` and you will see messages like:

```
root@vmf64:~# docker info
Client:
 Version:    24.0.6
 Context:    default
```

(continues on next page)

(continued from previous page)

```
Debug Mode: false
...
```

The following sections show how to work with Docker example images. For more details please refer to the official Docker documentation.

2.2.1 Get Docker images from the web

To start ready to use images from the Docker site you need one Ethernet adapter assigned to RT-Linux. This adapter needs to be configured for TCP/IP usage.

To pull (download) Docker image, you should start `docker pull hello-world`:

```
root@vmf64:~# docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53641cd209a4fecfc68e21a99871ce8c6920b2e7502df0a20671c6fccc73a7c6
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

Start the container via `docker run hello-world`. Then you will get messages like:

```
root@vmf64:~# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent␣
↪it
    to your terminal.
```

To get a list of locally available images, use `docker image ls`

```
root@vmf64:~# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    d2c94e258dcb   11 months ago   13.3kB
```

2.2.2 Use local docker images

This chapter shows how to work with local Docker images on RT-Linux. As an example we will pull (download) the BusyBox image from a global repository (docker.io). Then we will save and restore the image locally.

In a first step, get the docker image from the web (e.g. on a separate Linux PC).

```
root@vmf64:~# docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
7b2699543f22: Pull complete
Digest: sha256:650fd573e056b679a5110a70aabeb01e26b76e545ec4b9c70a9523f2dfaf18c6
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
```

Save the Docker image to the `busybox.tar` file:

```
root@vmf64:~# docker save busybox > busybox.tar
```

Next, copy the file `busybox.tar` to the guest shared folder, for example at `/hv/guests/` After you have made the image available at the guest shared folder you can use the `docker load` command to load the saved image into the local repository:

```
root@vmf64:~# docker load < /mnt/rtfiles/busybox.tar
95c4a60383f7: Loading layer [==================================================>
↪]  4.495MB/4.495MB
Loaded image: busybox:latest
```

Run the image:

```
root@vmf64:~# docker run -it busybox:latest
/ #
```

This will create and start a new container (`docker run`) in an interactive terminal (`-it`) by loading the image `busybox:latest` from the local repository. The interactive terminal will give you access to the BusyBox console. To exit the container use the `exit` command.

2.3 RT-Linux specific configuration parameters

Default settings are stored in the `guest.config` file in the guest folder. The user can override those setting in the `usr.config` file.

2.3.1 Network configuration

All network configuration parameters are stored in the `guest.config` file under the `[Rtos\Vnet\0]` key.

Entry Name	Type	Description
MacAddress	String	Virtual Network Adapter MAC address.
IpAddress	String	IP address for the RT-Linux virtual network adapter.
PollingPeriodMs	String	The polling period can be adjusted using this parameter. A value of 0 enables the interrupt mode.
PollingPeriodUs (optional)	dword	The period in usec activates high resolution polling mode. If absent or value of 0, uses PollingPeriodMs parameter.
NoNAPI (optional)	dword	The value of 1 disables NAPI for Vnet driver. May provide better performance when used with CODESYS

2.3.2 Autostart

Under the `[Rtos\Autostart\1]` key you can define optional autostart parameters.

Entry Name	Type	Description
Executable	String	Name and path of the executable file
Parameter	String	Parameters of the executable

In `guest.config` a default autostart script is configured. This script (`autostart.sh`) is located in the sub-folder `files` of the guest folder. You can redefine the name and location of the autostart script in `usr.config` as well as change the existing default settings in the shipped script.

The `[Rtos\Linux]` key contains Linux specific settings. The followings entries are optional and provide the possibility to disable some RT-Linux components:

Entry Name	Type	Description
novnet	String	If `yes` or `y`, VNET network will be not initialized
nossh	String	If `yes` or `y`, OpenBSD Secure Shell server will not be started
notelnet	String	If `yes` or `y`, Telnet server will not be started
nortosfs	String	If `yes` or `y`, real time file system will be not initialized and no mount for /mnt/rtfiles will be created
nortosservice	String	If `yes` or `y`, RTOS service will not be started

2.3.3 Kernel command line

The `[Rtos\Linux]` key can specify the kernel command line:

Entry Name	Type	Description
cmd_line	String	Command line string passed to the Kernel. Default value is nopti

2.3.4 Time synchronization settings

The `[Rtos\TimeSync]` key contains time synchronization settings. RT-Linux has here some own specific parameters. Please note, `[Rtos\TimeSync]` and `[Host\TimeSync]` can have other parameters as well, valid for Linux and all other operating systems. Please refer to `Hypervisor\doc\RtosVM-UserManual.pdf` to get details.

Entry Name	Type	Description
TimezoneLinux	String	Contains timezone for Linux, as in `TZ` Linux environment variable, ex. `Europe/Berlin`. By default, Linux starts with UTC timezone. All existing time zones could be found in `/usr/share/zoneinfo` directory. if this parameter is set and TaskEnabled parameter is not 0, boot script creates a link to the specified time zone binary file from `/etc/localtime`.
TaskEnabled	dword	0 Task will not be started 1 Task will be started 2 Task will be started, init time and timezone once and then finish. omitted = 0 = Task will not be started. It differs from omitted value from other OSes defined in `RtosVM User Manual`

2.4 Example Applications

2.4.1 General

To make your initial experiences in working with RT-Linux and Linux go smoothly, a number of example application programs have been provided with the product release. Some are intended as exercises to help familiarize you with various system features; others are useful tools. Much of the code can serve as model software that can save you time in developing your own applications.

The example applications are located at the product SDK. Please refer to *RT-Linux Guest SDK* to locate Linux or Windows SDK.

2.4.2 List of examples

These examples are based on the RTOS-Library. For more information about this library see the RTOS VM User Manual.

Windows	RT-Linux	Description
CSharpDemo		The C# demo shows you how to use RtosLib functions from a C# application.
EventDemo	EventDemo	Use the Shared event demo to experiment with Shared Events between OS.
	FileDemo	The File demo shows you how a Linux application can remotely access the RtFiles directory.
InterlockDemo	InterlockDemo	The interlock demo shows you how to use the Interlocked-CompareExchange function to synchronize shared memory access works between two OS.
MsgQueueDemo	MsgQueueDemo	Use the Message queue demo to see how a simple communication channel can be established between two OS using a shared memory based message queue to transfer data.
PipeDemo	PipeDemo	Use the Pipe demo to see how a simple communication channel can be established between two OS using a shared memory-based pipe to transfer data.
ShmDemo	ShmDemo	Use the Shared memory demo to see how a simple communication channel can be established between two OS using Shared Memory to transfer data.
SocketDemo	SocketDemo	Use the Socket demo to see how a simple communication channel can be established between two OS using shared memory-based socket to transfer data.
	TimerDemo	Use the Timer demo to see how a periodic task with real time priority can be started.

2.5 RT-Linux Guest SDK

The development of applications for RT-Linux Guest with the SDK can be done on Windows or Linux. This can involve using a Windows or Linux guest in the RTOSVisor or a separate PC with Windows or Linux.

Linux development:

The SDK required for developing RT-Linux applications on Linux can be found on the Hypervisor Host in the file `/hv/guests/files/LinuxTools/rtlinux-sdk.tar`. You can extract the SDK either directly within the Hypervisor Host or on a separate development PC running Linux. Currently the SDK includes three subdirectories: `/inc` for headers, `/lib` for libraries and `/examples` for sample applications. A Description for developing and debugging RT-Linux applications using Eclipse in Ubuntu is available in the Hypervisor Manual in section **remote-debugging:Debugging RT-Linux Guest with Eclipse on Ubuntu**. In addition, the source files to create the RT-Linux binaries are included (`sources_*.xz`).

Windows development:

The SDK required for developing RT-Linux applications on Windows is located in the file `Hypervisor\SDK\rtlinux-sdk.zip`, which is part of the RTOSVisor installation package. Instructions for developing and debugging RT-Linux applications using Microsoft Visual Studio in Windows are available in the section **remote-debugging:Debugging RT-Linux Guest with Visual Studio on Windows** in the Hypervisor Manual manual. A short description how to build the provided examples is included in the file `examples\linux\HowToBuildTheExamplesLinux.txt` in the SDK.

2.6 Real-time application programming

RT-Linux guest is shipped with a pre-configured and validated hard real-time capable Linux image.

The basic technology behind it is the `PREEMPT_RT` patch for Linux.

Valuable information about how to create real-time capable Linux applications can be found at the Linux Foundation WIKI: https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/start

2.7 RTOS Library

The RTOS library provides higher level communication services for synchronizing different operation systems or to exchange data between the operating systems. The RTOS library is based on VMF-functions which provide the basic communication functionality.

A description of the RTOS Library can be found in Chapter 10.1 of the `Hypervisor\doc\RtosVM-UserManual.pdf`, which is included in the RTOSVisor installation package.

2.8 TCP/IP configuration

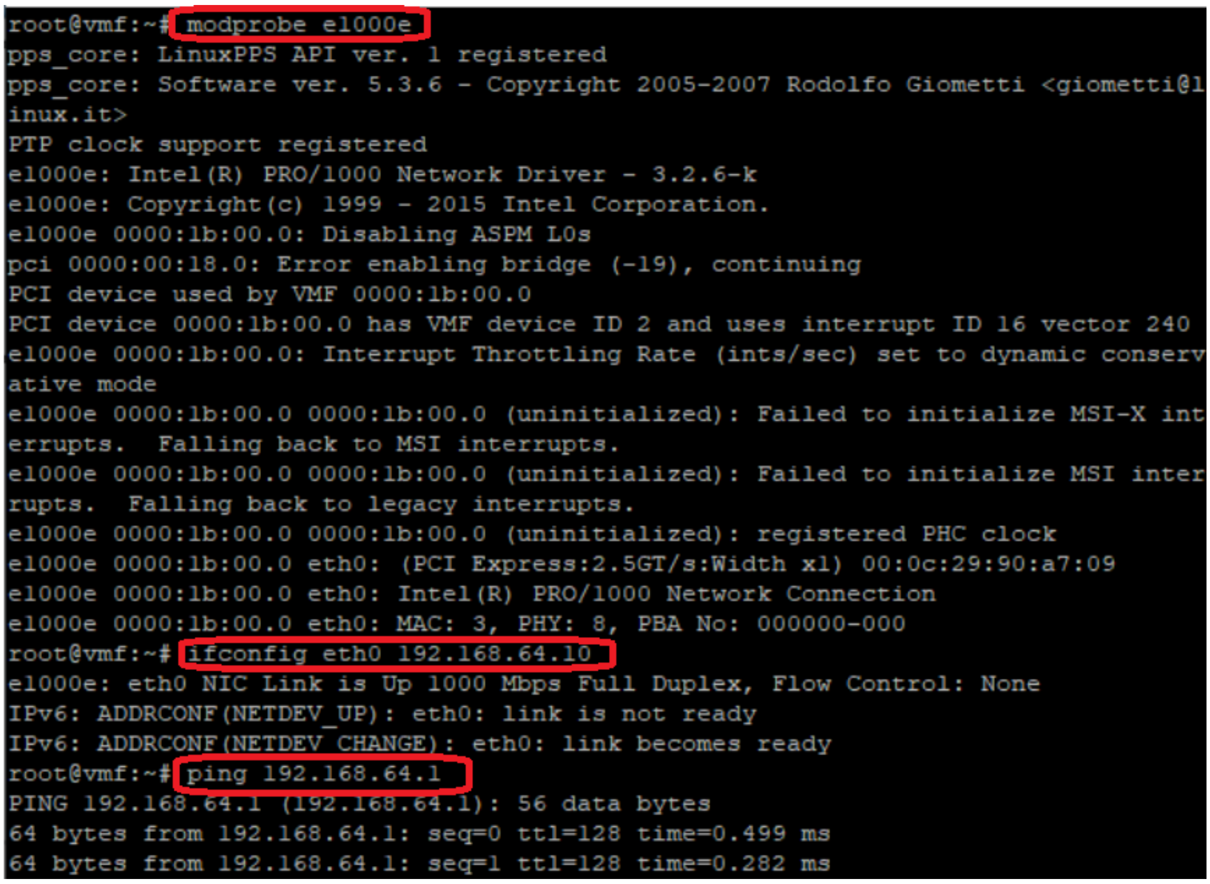
If you want to use TCP/IP in the guest, the appropriate drivers have to be loaded and the ethernet adapter must be configured appropriately.

Please uncomment the respective settings provided in the shipped autostart script.

```
# load Ethernet drivers if required
modprobe igb
modprobe igc
modprobe e1000e
modprobe r8169
ifconfig eth0 up

# set fixed IP address (note: the default kernel does not support DHCP)
ifconfig eth0 192.168.64.10
```

After restarting the guest, check if everything works: enter `ifconfig` and/or `ping` some device on the network.



```
root@vmf:~# modprobe e1000e
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@l
inux.it>
PTP clock support registered
e1000e: Intel(R) PRO/1000 Network Driver - 3.2.6-k
e1000e: Copyright(c) 1999 - 2015 Intel Corporation.
e1000e 0000:1b:00.0: Disabling ASPM L0s
pci 0000:00:18.0: Error enabling bridge (-19), continuing
PCI device used by VMF 0000:1b:00.0
PCI device 0000:1b:00.0 has VMF device ID 2 and uses interrupt ID 16 vector 240
e1000e 0000:1b:00.0: Interrupt Throttling Rate (ints/sec) set to dynamic conserv
ative mode
e1000e 0000:1b:00.0 0000:1b:00.0 (uninitialized): Failed to initialize MSI-X int
errupts.  Falling back to MSI interrupts.
e1000e 0000:1b:00.0 0000:1b:00.0 (uninitialized): Failed to initialize MSI inter
rupts.  Falling back to legacy interrupts.
e1000e 0000:1b:00.0 0000:1b:00.0 (uninitialized): registered PHC clock
e1000e 0000:1b:00.0 eth0: (PCI Express:2.5GT/s:Width x1) 00:0c:29:90:a7:09
e1000e 0000:1b:00.0 eth0: Intel(R) PRO/1000 Network Connection
e1000e 0000:1b:00.0 eth0: MAC: 3, PHY: 8, PBA No: 000000-000
root@vmf:~# ifconfig eth0 192.168.64.10
e1000e: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
root@vmf:~# ping 192.168.64.1
PING 192.168.64.1 (192.168.64.1): 56 data bytes
64 bytes from 192.168.64.1: seq=0 ttl=128 time=0.499 ms
64 bytes from 192.168.64.1: seq=1 ttl=128 time=0.282 ms
```

Important: If you are using the acontis EtherCAT master stack, you do not need these drivers, the master stack is using its own high performance drivers. You should only load these drivers if you need to use TCP/IP in parallel to EtherCAT. In this case, please check the EC-Master documentation for details, how to unbind the driver from the Ethernet MAC that is used for EtherCAT.

2.9 Access Windows file system using Samba

It is possible to access the file system of a Windows PC using Samba. Here is a description of how to start samba automatically via a start-up shell-script.

1. Create a new Windows share in Windows or use existing one.
2. Modify the `usr.config` file by adding the following two lines. This will prevent the default `autostart.sh` script from being called. Alternatively, you can skip replacing the autostart script and add the call to `cifs_mnt.sh` directly in `autostart.sh`.

```
[Rtos\Autostart\1]
"Executable"="/mnt/rtfiles/cifs_mnt.sh"
```

3. Create a new file named `cifs_mnt.sh` at `/hv/guests/` with the following content:

```
HOST_IP="192.168.157.1"
SHARE_NAME="SomeName"
```

(continues on next page)

(continued from previous page)

```
USER="UserName"
PASSWORD="UserPassword"
mkdir -p /mnt/${SHARE_NAME}
mount -t cifs //${HOST_IP}/${SHARE_NAME} /mnt/${SHARE_NAME} -v -o
username=${USER},password=${PASSWORD},domain=${HOST_IP},vers=2.1
echo "cifs mounted"
```

4. Modify the `HOST_IP`, `SHARE_NAME`, `USER` and `PASSWORD` parameters appropriate to your windows share from step 1.
5. Take care that you create the `cifs_mnt.sh` file with `LF` (Line Feed) line endings. Windows programs create `CR/LF` line ending, by default. Shell pre-processor cannot recognize this so you have to convert the line endings to only `LF`. Common Editors have functions to do that.
6. Start RT-Linux and verify the mount results with the `mount` command.

2.10 Remote Debugging

See the respective chapter in the Hypervisor Manual.

2.11 Synchronize Linux Time with Host

RT-Linux contains a special time synchronisation daemon, that works in background and synchronizes Linux time with host time. This daemon is disabled by default, but it starts automatically when the time synchronization feature is enabled in the configuration.

To enable this feature modify the following config section in `/hv/config/hvbase.config`:

```
[Host\TimeSync]
"TimeSyncMaster"="Host"
"TaskEnabled"=dword:1
```

And add or modify the following config section in `/hv/guests/guestxxxx/usr.config`

```
[Rtos\TimeSync]
"TimeSyncMaster"="Host"
"TaskEnabled"=dword:1
"TimezoneLinux"="Europe/Berlin"
```

Now start RT-Linux and check for a new log message showing that the timesync daemon has been started If you want to have different timezone in Linux and Host, you should set TZ variable in autostart script.

All existing time zones could be found in `/etc/share/timezone` directory. Default RT-Linux image does not contain all available timezones due to size considerations. If need to use other timezones, please change `yocto/recipes-core/images/rtlinux-image-initramfs.bb` file.

2.12 How to create your own RT-Linux Image (BSP)

See *How to create your own RT-Linux Image (BSP)*

2.13 RT-Linux GPL compliance

RT-Linux uses open-source packets with GPL licenses. RT-Linux is compliant to GPL V2, because the Linux kernel and the BusyBox (user interface shell including several commands) are based on GPL V2.

Compliance to the FLOSS (Free/Libre Open Source Software) licenses are achieved in RT-Linux by the following means:

- The related source code is provided.
- License text for the used software is provided.
- Compilation scripts and modifications to the source code are provided.

RT-Linux source code and compilation scripts could be found inside *RT-Linux Guest SDK*.

All licenses texts are located at `/hv/doc/licenses/rtlinux`:

Important: Customer software which is added into the Linux **kernel** of RT-Linux also will have to be GPL compliant according to the above rules.

Customer software and applications running in **user mode** has not to be compliant to GPL and thus can stay closed source and can be provided under any kind of software license the customer chooses.

---

3 Windows Subsystem for Linux

As long as you do not need deterministic real-time behavior, you may use WSL2 (Windows Subsystem for Linux version 2) for development purposes of your Real-time Linux applications. Physical hardware access is also rather limited when using WSL2.

3.1 Setting Up WSL2 and Ubuntu 24.04 on Windows 10

This guide provides step-by-step instructions on how to enable Windows Subsystem for Linux (WSL2) and install Ubuntu 24.04 on a Windows 10 system.

3.1.1 Prerequisites

- A system running **Windows 10**, version 1903 or higher.
- Internet connection to download the necessary files from Microsoft Store.

3.1.2 Step 1: Enable Windows Subsystem for Linux (WSL)

1. **Open PowerShell as Administrator**: Right-click on the Start button and select **Windows PowerShell (Admin)**.
2. **Enable WSL**: In the PowerShell window, run the following command to enable WSL:

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-
↪Subsystem-Linux /all /norestart
```

3. **Enable Virtual Machine Platform**: WSL2 requires the Virtual Machine Platform feature. Run the following command to enable it:

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /
↪all /norestart
```

4. **Restart Your Computer**: After enabling both features, restart your machine to apply the changes.

3.1.3 Step 2: Set WSL2 as the Default Version

1. **Open PowerShell as Administrator**.
2. **Set WSL2 as the Default Version**: Run the following command to set WSL2 as the default version for any future Linux distributions:

```
wsl --set-default-version 2
```

3.1.4 Step 3: Install Ubuntu 24.04

1. **Open Microsoft Store**: Open the **Microsoft Store** from the Start menu.
2. **Search for Ubuntu**: Search for **Ubuntu 24.04** in the Microsoft Store.
3. **Install Ubuntu 24.04**: Click the **Install** button to download and install Ubuntu 24.04.

3.1.5 Step 4: Initialize Ubuntu 24.04

1. **Launch Ubuntu**: Open **Ubuntu 24.04** from the Start menu or by typing "Ubuntu" in the search bar.
2. **Set Up Ubuntu**: On the first launch, follow the on-screen instructions to create a new user and set a password.
3. **Update Packages**: After the setup, it is recommended to update the package list and upgrade any outdated packages:

```
sudo apt update
sudo apt upgrade -y
```

3.1.6 Optional: Check WSL Version

You can verify the installed WSL distributions and their versions, or set a specific version for a distribution.

- **List installed WSL distributions and versions**:

```
wsl -l -v
```

- **Set WSL version for a specific distribution**:

```
   wsl --set-version <distro_name> 2

Replace ``<distro_name>`` with the name of the installed distribution␣
↪(e.g., ``Ubuntu-24.04``).
```

After following these steps, you should have WSL2 and Ubuntu 24.04 installed and running on your Windows 10 machine.

3.2 Enabling SSH on Ubuntu 24.04 in WSL2

This guide will walk you through enabling and configuring SSH on an Ubuntu 24.04 distribution running in WSL2 on Windows 10.

3.2.1 Step 1: Install the OpenSSH Server

1. **Open your Ubuntu terminal** in WSL2.
2. **Install the OpenSSH server** by running the following commands:

```
sudo apt update
sudo apt install openssh-server -y
```

3.2.2 Step 2: Start and Enable the SSH Service

1. **Start the SSH service** by running:

```
sudo service ssh start
```

2. **Check the SSH service status** to ensure it is running:

```
sudo service ssh status
```

3. (Optional) **Enable SSH service on startup**. While WSL2 doesn't start services automatically by default, you can enable SSH to start automatically if needed:

```
sudo systemctl enable ssh
```

3.2.3 Step 3: Configure SSH (Optional)

1. **Edit the SSH configuration file** located at:

```
sudo nano /etc/ssh/sshd_config
```

2. Modify the settings as needed, such as changing the default port or enabling/disabling root login.
3. After making any changes, **restart the SSH service** to apply them:

```
sudo service ssh restart
```

3.2.4 Step 4: Find the WSL2 IP Address

WSL2 uses dynamic IP addresses that change with each session. To connect via SSH, you need the current IP address of your WSL2 instance.

1. **Find the WSL2 IP address** by running:

```
ip addr show eth0 | grep inet
```

2. The command will output something like `inet 172.30.32.1/20`. The IP address before the slash (e.g., `172.30.32.1`) is the address you will use to connect via SSH.

3.2.5 Step 5: Connect via SSH from Windows

1. **From the Windows host**, you can use an SSH client, such as PowerShell or PuTTY.
2. **Connect to your WSL2 instance** from a PowerShell terminal by running:

```
ssh username@<wsl_ip_address>
```

   Replace `username` with your Ubuntu username and `<wsl_ip_address>` with the IP address you retrieved in Step 4.

3.2.6 Step 6: Enable Port Forwarding (Optional)

If you want to access the SSH server from outside WSL2 (e.g., from Windows or external devices), set up port forwarding:

1. **Open PowerShell as Administrator** and run the following command:

```
netsh interface portproxy add v4tov4 listenport=2222 listenaddress=0.
↪0.0.0 connectport=22 connectaddress=<wsl_ip_address>
```

   Replace `<wsl_ip_address>` with the IP address of your WSL2 instance.

2. **SSH into your WSL2 instance** from the Windows host using:

```
ssh username@localhost -p 2222
```

   Replace `username` with your Ubuntu username.

3.3 Setting Up Avahi Daemon for mDNS with a Fixed Hostname

This guide explains how to install *avahi-daemon* on Ubuntu in WSL2 and configure a fixed hostname for mDNS.

3.3.1 Step 1: Install Avahi Daemon

1. **Open the Ubuntu terminal** in WSL2.
2. **Install `avahi-daemon`** by running the following commands:

```
sudo apt update
sudo apt install avahi-daemon -y
sudo service avahi-daemon start
```

3.3.2 Step 2: Check the mDNS hostname

1. Check the mDNS hostname:

   ```
   systemctl status avahi-daemon
   ```

   The avahi daemon may have recognized a hostname conflict with Windows 10 and thus chose another name. This name typically ends with .local.

3.3.3 Step 3: Verify the New Hostname

Ping the new `.local` hostname to verify mDNS is working:

```
ping newhostname.local
```

3.4 Setting Up Remote Debugging

3.4.1 Step 1: Install gdbserver

1. **Open the Ubuntu terminal** in WSL2.
2. **Install `gdbserver`** by running the following commands:

   ```
   sudo apt update
   sudo apt install gdbserver -y
   ```

3.4.2 Step 2: Configure Debugging Client (e.g. Visual Studio with VisualGDB)

1. **Open the Debug Client**, e.g. Visual Studio
2. **Configure the SSH connection** to use the IP address or the hostname of Ubuntu.

4 Docker Container

Container offer a powerful way to package applications along with their dependencies, creating lightweight and portable units that can run consistently across a wide variety of environments. Whether on a developer's local machine, in a test environment, or on a production server, containers ensure that the application behaves the same everywhere.

One of the key advantages of containers is the isolation they provide. Applications running within a container are completely self-contained, with all necessary libraries and dependencies bundled inside. This eliminates conflicts with other software on the host system and guarantees consistent behavior regardless of the underlying infrastructure.

In addition to their portability and isolation, containers provide robust version control capabilities. docker images, which are the building blocks of containers, can be versioned. This allows developers to easily roll back to previous versions if an update causes issues or to update containers with minimal effort, ensuring smooth transitions between software versions.

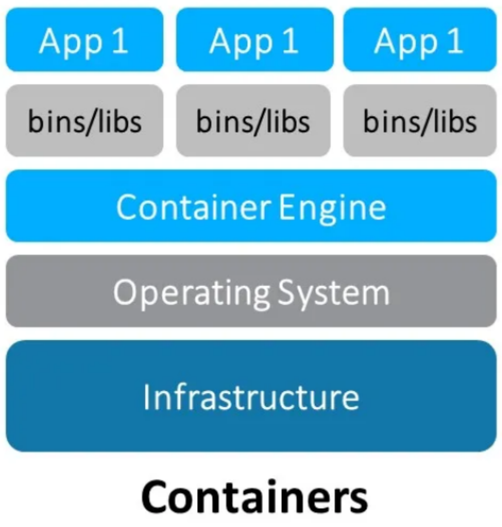


Fig. 4.1: Container Architecture

See chapter *Example Real-time Linux guest with Docker* how to select a RT-Linux image with docker container support. This chapter also shows how to work with docker in general.

4.1 Docker Container Repository (in general)

A **docker container repository** is a storage location where docker images are stored, either locally on a system or remotely on a platform such as **docker Hub**, **Google Container Registry (GCR)**, **Amazon Elastic Container Registry (ECR)**, or a private docker registry. docker images contain all the necessary code, libraries, and dependencies required to run an application inside a container.

4.1.1 Key Aspects of a Docker Container Repository

1. **Repository vs. Image**:
   - A **docker image** is a snapshot of an application and its dependencies, essentially a template for creating containers.
   - A **docker repository** is a collection of related docker images, often versions of the same application. For example, `myapp:1.0`, `myapp:2.0`, and `myapp:latest` are different images in the same repository.
2. **Local Repository (On your machine)**:
   - docker images are first built and stored locally on the user's machine. This is often referred to as the **local docker repository**.
   - To see the images stored locally, you can run:

     ```
     docker images
     ```

     This command lists all the images in your local repository, showing the **repository name**, **tags** (versions), **image ID**, and **size**.
3. **Remote Repository**:
   - Remote repositories are hosted on platforms like **docker Hub**, **AWS ECR**, **GCR**, or any private docker registry.
   - These repositories store and distribute images so that users can `push` and `pull` docker images as needed. When you run `docker pull` to download an image or `docker push` to upload an image, you are interacting with a remote repository.
4. **Repository Contents**: A docker repository contains the following:
   - **Images**: Each repository contains one or more **docker images**. Each image can have multiple **tags**, which identify different versions or configurations of the same image. For example, a repository for `myapp` may contain images with tags like `latest`, `v1.0`, `v2.0`, etc.
   - **Image Layers**: docker images are made up of layers, where each layer represents a change or update (such as installing a library or changing a file). Layers allow docker to optimize storage and download by only pulling layers that are not already available locally.
   - **Metadata**: Each image comes with metadata that includes details about its creation, such as the image size, the commands used to create it (e.g., from the Dockerfile), and environment variables that might be used when running containers from the image.
5. **Private vs. Public Repositories**:
   - **Public repositories** allow anyone to pull and use images. docker Hub is an example of a public registry where popular open-source images are available (e.g., Ubuntu, MySQL).
   - **Private repositories** restrict access, often used in enterprise environments to store proprietary or internal application images.

4.1.2 Basic Commands for Interacting with a Docker Repository

- **Listing Images in Local Repository**:

  ```
  docker images
  ```

  This shows the images currently stored on your machine.

- **Pulling an Image from a Remote Repository**:

  ```
  docker pull <repository>:<tag>
  ```

  Example:

  ```
  docker pull ubuntu:latest
  ```

- **Pushing an Image to a Remote Repository**:

  Before pushing an image, you must first log in to the repository (if it's private):

  ```
  docker login
  ```

  Then, you can push the image:

  ```
  docker push <repository>:<tag>
  ```

  Example:

  ```
  docker push myrepo/myapp:1.0
  ```

4.1.3 Example of a Docker Repository and Its Contents

For example, in **docker Hub**, if you have a repository named `myapp`, it may contain the following images:

- `myapp:latest`: The latest stable version of the app.
- `myapp:v1.0`: The first major release of the app.
- `myapp:v2.0`: The second major release of the app, perhaps with significant feature changes or improvements.

Each image will have its own layers and metadata, and you can pull specific versions by specifying the tag (`:latest`, `:v1.0`, etc.).

In summary, the **docker container repository** serves as a centralized store for docker images, enabling easy sharing, versioning, and distribution of containerized applications across environments.

4.2 Local RT-Linux Docker Container Repository

When using docker container in RT-Linux, docker images will be stored either locally in RAM or in a persistent repository.

The persistent repository will be stored onto a filesystem which is located in a raw file image created by the `qemu-img` tool at the very first start of the RT-Linux guest. It will only be created if the parameter `rt_data_size_MB` is set in the `guest_config.sh` or `usr_guest_config.sh` guest configuration files.

The RT-Linux autostart script then calls the `rtdata_init.sh` script located in the guest folder. This script will mount the filesystem image at `/mnt/rtdata` when RT-Linux boots.

After mounting the filesystem the `/mnt/rtfiles/etc/rt-linux/files/docker-env/dockerenv_init.sh` script will be called which will mount the docker repository into `/mnt/rtdata`. These mountpoints (which then will be located in the persistent filesystem `/mnt/rtdata`) are defined in `/hv/guests/etc/rt-linux/files/docker-env/dockerenv_dirs.sh`.

After starting a RT-Linux guest with docker support, you can verify which filesystems are mounted at the persistent file image:

```
root@vmf64:~# mount | grep loop0
/dev/loop0 on /mnt/rtdata type ext4 (rw,relatime)
/dev/loop0 on /var/lib/docker type ext4 (rw,relatime)
/dev/loop0 on /var/opt/codesysvcontrol type ext4 (rw,relatime)
/dev/loop0 on /home/root type ext4 (rw,relatime)
/dev/loop0 on /root type ext4 (rw,relatime)
```

See chapter *Example Real-time Linux guest with Docker* how to select a RT-Linux image with docker support and how to work with it.

4.3 Docker Base Image

A **docker base image** is the foundational layer upon which all other layers in a docker container are built. It's essentially the starting point for creating a docker image, and it doesn't inherit from any other image.

A base image can either be:

1. **A minimal operating system** such as `Ubuntu`, `Alpine`, or `Debian`, which provides a basic environment to run applications.
2. **An empty image** (sometimes referred to as a scratch image), which allows you to build everything from scratch without any pre-existing layers.

4.3.1 Key Concepts of Docker Base Images

1. **Predefined vs. Custom Base Images**:
   - **Predefined Base Images**: These are popular base images created and maintained by communities or organizations. For example:
     - `ubuntu`: A full-fledged operating system image based on Ubuntu.
     - `alpine`: A very small and lightweight Linux distribution.
     - `node`, `python`: Images that not only include an operating system but also come pre-configured with specific programming languages or runtimes.
   - **Custom Base Images**: You can create a custom base image by starting from a minimal OS or from scratch, and then adding your specific dependencies or configuration.
2. **How to Use a Base Image**: When writing a `Dockerfile` to build an image, you typically specify the base image using the `FROM` instruction:

```
# Example of using a base image (Ubuntu)
FROM ubuntu:20.04

# Commands to install your application and dependencies
RUN apt-get update && apt-get install -y nginx
CMD ["nginx", "-g", "daemon off;"]
```

   In this example, `ubuntu:20.04` is the base image. All subsequent layers are built on top of this.
3. **Why Use a Base Image?**:
   - **Simplicity**: A base image provides a ready-made environment, so you don't have to start from scratch and set up the entire OS yourself.
   - **Standardization**: Base images offer a standardized foundation, ensuring that everyone using the image gets a consistent environment.
   - **Efficiency**: Many base images are lightweight, allowing for efficient use of resources and smaller image sizes. For example, the `Alpine` base image is just a few megabytes in size compared to other larger operating systems like `Ubuntu` or `CentOS`.
4. **Popular docker Base Images**:
   - **Alpine**: A tiny Linux distribution known for its very small size (~5 MB) and simplicity. It's used for minimalist environments.

```
FROM alpine
```

   - **Ubuntu**: One of the most commonly used base images, providing a complete Linux OS.

```
FROM ubuntu:20.04
```

   - **Debian**: A stable and widely-used Linux distribution.

```
FROM debian:buster
```

   - **Node.js, Python, Java**: These images come pre-configured with popular programming languages or runtimes, allowing developers to build applications with the necessary software pre-installed.

```
FROM node:14
```

```
FROM python:3.8
```

5. **Scratch Base Image**:

   - `scratch` is a special base image in docker that represents an empty image. It is useful when you want to create extremely lightweight images that contain only the application itself with no operating system files.
   - Commonly used when building Go, C, or C++ applications that don't require an entire operating system for runtime.

   ```
   FROM scratch
   COPY my-binary /my-binary
   CMD ["/my-binary"]
   ```

6. **Creating Your Own Base Image**:

   If you have specific needs that are not met by an existing base image, you can create your own:

   - You can start from a minimal base image (like `alpine`) and add custom tools and configurations.
   - Once your custom base image is ready, you can push it to a docker registry and reuse it as a base for other Dockerfiles.

4.3.2 When to Choose Different Base Images

- **Small Application Footprint**: If you need the smallest possible image, consider using `alpine` or even `scratch`.
- **Specific Environment**: If your application depends on a specific version of a language or OS, choose a base image that matches your requirements, like `node`, `python`, or `ubuntu`.
- **Custom Needs**: If none of the available base images fit your needs, you can create your own by starting from a minimal OS or using `scratch`.

4.4 Set up Docker

4.4.1 Docker on Ubuntu

Create a script `setupdocker.sh` on Ubuntu to install the docker environment:

```
#!/bin/bash

# Exit on error
set -e

# 1. Update the package list
echo "Updating package list..."
sudo apt-get update

```

(continues on next page)

(continued from previous page)

```
# 2. Install dependencies for APT to use HTTPS
echo "Installing necessary dependencies..."
sudo apt-get install -y ca-certificates curl gnupg lsb-release

# 3. Add docker's official GPG key
echo "Adding docker's official GPG key..."
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
↪dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# 4. Set up the stable docker repository for Ubuntu
echo "Setting up docker repository..."
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
↪docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu
↪$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.
↪list > /dev/null

# 5. Update the package list again
echo "Updating package list after adding docker repository..."
sudo apt-get update

# 6. Install docker Engine, CLI, and plugins
echo "Installing docker Engine and other components..."
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-
↪buildx-plugin docker-compose-plugin

# 7. Start and enable docker
echo "Starting and enabling docker service..."
sudo systemctl start docker
sudo systemctl enable docker

# 8. Verify docker installation
echo "Verifying docker installation..."
docker_version=$(sudo docker --version)
echo "docker has been installed: $docker_version"

# 9. Test docker installation
echo "Running docker hello-world test..."
sudo docker run hello-world

# Optional: Add the current user to the docker group
echo "Would you like to run docker without sudo? (y/n)"
read -r add_to_group

if [ "$add_to_group" = "y" ]; then
    sudo usermod -aG docker $USER
    echo "Added $USER to the docker group. Please log out and log back in␣
↪or run 'newgrp docker' to apply changes."
else
    echo "You can still run docker with sudo."
fi

echo "docker installation is complete."
```

Run the script to install docker:

```
$ chmod +x ./setupdocker.sh
$ ./setupdocker.sh
```

4.4.2 Docker on Debian

Create a script `setupdocker.sh` on Debian to install the docker environment:

```
#!/bin/bash

# Exit on error
set -e

# 1. Update the package list
echo "Updating package list..."
sudo apt-get update

# 2. Install dependencies for APT to use HTTPS
echo "Installing necessary dependencies..."
sudo apt-get install -y ca-certificates curl gnupg lsb-release

# 3. Add docker's official GPG key
echo "Adding docker's official GPG key..."
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --
↪dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# 4. Set up the stable docker repository for Debian
echo "Setting up docker repository..."
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
↪docker-archive-keyring.gpg] https://download.docker.com/linux/debian
↪$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.
↪list > /dev/null

# 5. Update the package list again
echo "Updating package list after adding docker repository..."
sudo apt-get update

# 6. Install docker Engine, CLI, and plugins
echo "Installing docker Engine and other components..."
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-
↪buildx-plugin docker-compose-plugin

# 7. Start and enable docker
echo "Starting and enabling docker service..."
sudo systemctl start docker
sudo systemctl enable docker

# 8. Verify docker installation
echo "Verifying docker installation..."
docker_version=$(sudo docker --version)
echo "docker has been installed: $docker_version"

# 9. Test docker installation
echo "Running docker hello-world test..."
sudo docker run hello-world

# Optional: Add the current user to the docker group
echo "Would you like to run docker without sudo? (y/n)"
read -r add_to_group

if [ "$add_to_group" = "y" ]; then
```

(continues on next page)

(continued from previous page)

```
    sudo usermod -aG docker $USER
    echo "Added $USER to the docker group. Please log out and log back in␣
↪or run 'newgrp docker' to apply changes."
else
    echo "You can still run docker with sudo."
fi

echo "docker installation is complete."
```

Run the script to install docker:

```
$ chmod +x ./setupdocker.sh
$ ./setupdocker.sh
```

4.4.3 Windows 10 (WSL2)

- Enable WSL2 and install Ubuntu. See *Setting Up WSL2 and Ubuntu 24.04 on Windows 10*
- Install docker Desktop (https://www.docker.com/products/docker-desktop/) and enable WSL2 integration
- Install docker support: *Docker on Ubuntu*

4.5 Example: Yocto RT-Linux docker base image

In this example a docker base image will be created based on the Yocto RT-Linux guest image.

4.5.1 Base image creation

See chapter *Docker Base Image* for background information about docker base images.

To create such base image, we need to extract the root filesystem from the Yocto RT-Linux image.

For this purpose you will have to create a RT-Linux guest with no docker support in a first step. For example create a RT-Linux guest using the System Manager and select the `Linux 5.15 x64` image. After starting the guest, you need to log in and run the following command (the folders may be different on your RT-Linux image!):

```
$ tar -czf /mnt/rtfiles/files/rootfs.tar.gz /bin /conf /etc /init /lib64
↪  /root /sbin /usr /boot /home /lib /linuxrc >/dev/null
```

The `rootfs.tar.gz` will be located on the Hypervisor host at `/hv/guests/files`. Copy this image to your build machine, e.g. into `~/docker/baseimage` and create a new file `Dockerfile` in this folder with the following content:

```
FROM scratch
ADD rootfs.tar.gz /
CMD ["/bin/sh"]
```

Create the base image:

```
$ sudo docker build -t yocto-base-image ./
```

Test if the image exists and works:

```
$ sudo docker images
```

The newly created image should be listed.

```
$ sudo docker run -it yocto-base-image /bin/sh
```

You should see a shell prompt and will be able to type in typical shell commands. To leave the container, type `exit`.

Finally, you may save the base image for later usage.

```
$ sudo docker save -o yocto-base-image.tar yocto-base-image
```

4.6 Example: docker container for the EC-Master demo

The Yocto RT-Linux base image from the foregoing example will be used as the base platform for the EtherCAT master stack demo application.

4.6.1 Create an EC-Master demo container

Create a new folder where to create this container, e.g. `~/docker/ecm`.

Copy all binary files which are needed into this folder.

Finally, create a new docker configuration file `ecm.cfg` in this folder with the following content:

```
# Use your Yocto-based image as the base
FROM yocto-base-image:latest

# Create directories for applications and shared libraries
RUN mkdir -p /usr/local/bin /usr/local/lib

# Copy applications into the container
COPY EcMasterDemo /usr/local/bin/EcMasterDemo
COPY EcMasterDemoSyncSm /usr/local/bin/EcMasterDemoSyncSm

# Set execute permissions on the applications
RUN chmod +x /usr/local/bin/EcMasterDemo /usr/local/bin/EcMasterDemoSyncSm

# Copy shared libraries into the container
COPY libemllSockRaw.so /usr/local/lib/libemllSockRaw.so
COPY libemllIntelGbe.so /usr/local/lib/libemllIntelGbe.so
COPY libEcMaster.so /usr/local/lib/libEcMaster.so

# Initialize LD_LIBRARY_PATH
ENV LD_LIBRARY_PATH="/usr/local/lib:/lib:/lib64"

# Choose the application to run by default when the container starts
CMD ["/usr/local/bin/EcMasterDemo", "-sockraw", "eth0", "1"]
```

Create the container:

```
$ sudo docker build -t yocto-ecm-container -f ecm.cfg ./
```

Test if the image exists and works:

```
$ sudo docker images
```

Next, save the base image.

```
$ sudo docker save -o yocto-ecm-container.tar yocto-ecm-container
```

4.6.2 Run the EC-Master demo container

The `yocto-ecm-container.tar` file which you have created must be transferred now to the Hypervisor Host. You may copy it at `/hv/guests/files`.

Create a RT-Linux guests with docker support and start this guest.

Load the container:

```
$ docker load </mnt/rtfiles/files/yocto-ecm-container.tar
```

Run the container:

```
$ docker run --privileged -it yocto-ecm-container
```

You should see the master demo application start running. You may have to adjust the commandline parameters in `ecm.cfg` according to your needs.

5 Yocto Real-time Linux

5.1 Introduction to the Yocto Project

The **Yocto Project** is an open-source framework that provides a robust environment for creating custom Linux distributions. Unlike general-purpose Linux distributions, Yocto is designed to build a Linux operating system tailored specifically to your hardware and use case, ensuring the system includes only the components you need.

With **Yocto Linux**, you can develop a custom-built operating system that is optimized for your hardware, whether it's for embedded systems, IoT devices, or other specialized platforms. The Yocto Project offers a range of tools to configure and build the OS, allowing developers to strip away unnecessary software, thereby reducing the overall size and complexity of the distribution.

Key Benefits of Yocto

- **Customizability**: Yocto provides granular control over the operating system's composition. You can include only the essential components required for your project, minimizing bloat and reducing attack surfaces.
- **Smaller Footprint**: By tailoring the OS to specific needs, **Yocto Linux** results in a highly efficient and lightweight distribution. This makes it particularly well-suited for resource-constrained environments such as embedded systems, where memory and storage are limited.
- **Flexibility**: The Yocto Project offers full control over every aspect of the build process. This includes the selection of the Linux kernel, libraries, and user space applications. Developers can create and maintain their own board support packages (BSPs) and toolchains to ensure the OS is fine-tuned for the intended hardware platform.

5.2 How to create your own RT-Linux Image (BSP)

The Real-time Linux guest image is created using Yocto. The *RT-Linux Guest SDK* is shipped with several Yocto source packages:

- `sources_x86*.tar.xz` - sources for 32-bit Linux version
- `sources_x64*.tar.xz` - sources for 64-bit Linux version

Each packet contains sources and closed source binaries required to build the respective RT-Linux image.

5.2.1 Yocto Build System Installation

Linux

It is recommended to use Ubuntu 22.04 or newer to build the RT-Linux image. Copy the `sources_x*.tar.xz` file that is related to the RT-Linux image you want to create to your Linux machine and decompress it with the following command, e.g. into folder `~/yocto/rt_linux`:

```
$ tar -xf sources_x*.tar.xz
```

After decompression the script `prepare.sh` is available.

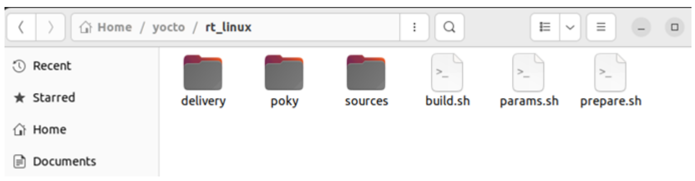


Fig. 5.1: Yocto root folder

Run the script to install all required packages.

```
$ sudo ./prepare.sh
```

Docker

You may run the yocto build in a Docker container. In a first step, a container with Yocto support must be generated.

Create the `yoctobld.cfg` file with the following docker configuration:

```
# Dockerfile for yocto build container

# Use a lightweight Ubuntu base image
FROM ubuntu:22.04

# Set environment to noninteractive to avoid prompts during build
ENV DEBIAN_FRONTEND=noninteractive

# Add i386 architecture for installing 32-bit packages
RUN dpkg --add-architecture i386

# Update package lists, install minimal dependencies, and remove apt caches
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
       ca-certificates \
       curl \
       netbase \
       git-core diffstat unzip texinfo gcc-multilib build-essential socat \
           cpio python3 python3-pip python3-pexpect python3-git python3-
↪jinja2 python3-venv \
           python3-dev python3-distutils xz-utils debianutils iputils-ping␣
↪libsdl1.2-dev \
           xterm libssl-dev lz4 liblz4-tool make diffstat chrpath zstd \
           p7zip-full g++-multilib libelf-dev file gawk wget \
           zstd libfuse-dev:i386 \
           locales \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Set up en_US.UTF-8 locale
RUN apt-get update && apt-get install -y locales \
    && echo "en_US.UTF-8 UTF-8" > /etc/locale.gen \
    && locale-gen en_US.UTF-8
```

(continues on next page)

(continued from previous page)

```
# Create user myuser with password MyUserPassword123!
RUN useradd -m myuser \
    && echo "myuser:MyUserPassword123!" | chpasswd \
    && usermod -aG sudo myuser

# Set environment variables for locale
ENV LANG=en_US.UTF-8 \
    LC_ALL=en_US.UTF-8

# default command
# Run a simple Python HTTP server
CMD ["/bin/bash"]
```

Then create the `blddocker.sh` script:

```
#!/bin/bash

# Name of the Docker image
IMAGE_NAME="yoctobld"

# Pull the latest Ubuntu image (optional)
echo "Pulling the latest Ubuntu base image..."
docker pull ubuntu:22.04

# Build the Docker image
echo "Building the yoctobld docker image..."
docker build --no-cache -f yoctobld.cfg -t $IMAGE_NAME .
```

Create the container:

```
$ chmod +x blddocker.sh
$ ./blddocker.sh
```

Create the docker start script `rundocker.sh`:

```
#!/bin/bash
docker run -it --user myuser --rm \
    --workdir=/rt_linux \
    -v ~/docker/yoctobld:/rt_linux \
    -v ~/docker/yoctobld:/home/$(whoami)/docker/yoctobld \
    -v ~/docker/yoctobld/rte:/home/rte \
    yoctobld /bin/bash
```

Start the container:

```
$ chmod +x rundocker.sh
$ ./rundocker.sh
```

Windows 10 (WSL2)

You can use the docker container from the previous chapter to set up the yocto build environment in Windows 10.

See *Windows 10 (WSL2)* on how to install Docker in Windows 10.

5.2.2 Build the RT-Linux Image

By default, all required packages are downloaded from the Internet. If you want to speed up the build, you may copy the `linux-stable-rt.git` repository (to be requested from your software supplier) into the folder `$bldroot\rte\p` where `$bldroot` is the root folder of the yocto build.

Run the `build.sh` script to build the image.

```
$ sudo ./build.sh
```

The resulting binary file will be stored in `delivery/Bin/rtlinux.bin`.

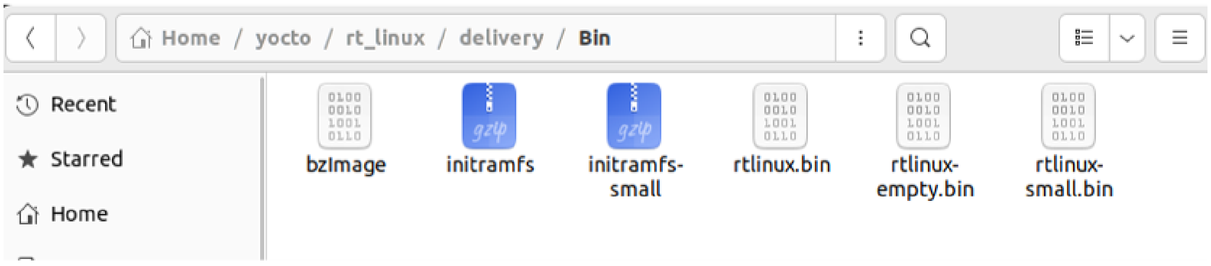


Fig. 5.2: Yocto delivery folder

5.2.3 How to change the default RT-Linux Image

Yocto is used to build the images. Yocto is a flexible and well documented system. If the shipped default image does not satisfy your requirements, you can change it.

Here is the description of some files which are likely candidates for changes:

- `sources/yocto/recipes-core/images/rtlinux-image-initramfs.bb` - BitBake file which describes all packages included to the default image
- `sources/yocto/recipes-core/images/rtlinux-image-initramfs-small.bb` - BitBake file for a minimal image. Can be used for systems with low memory footprint or as an example that shows which packages could be removed.

Both of these Bitbake files change the `PACKAGE_INSTALL` variable. This variable shows which packages will be installed on the system.

`sources/yocto/recipes-kernel/linux/linux-yocto-rt/defconfig` and `sources/yocto/recipes-kernel/linux/linux-yocto-rt/x86-64/defconfig` are Kernel configuration files for 32- and 64-bit versions.

Possible use cases when you may need to adjust the RT-Linux image:

1. Reduce the size of the image.

- You can remove packages from the `PACKAGE_INSTALL` variable. For example, GDB server is not required in the production image.

- You can install only required kernel modules instead of all modules. Package `kernel-modules` installs all modules. Inside the `rtlinux-image-initramfs-small.bb` file you can see how to install specific modules.

2. Add specific closed source package.

- Package `sources/yocto/recipes-core/initrdscripts` contains a list of files added to the image. You can see several binary files are located inside `initrdscripts/files/usr/bin` directory.

3. Change the Kernel configuration.

- defconfig is the Kernel configuration file.

4. Change default user credentials:

- By default, the user `root` with the password `root` is used. It is defined in the `sources/yocto/conf/layer.conf` file.

After you have made changes to the Yocto configuration files, run the build script.

```
$ ./build.sh
```

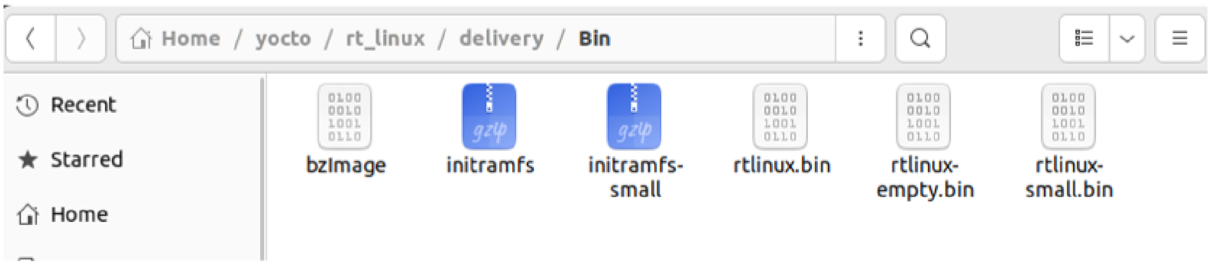


Fig. 5.3: Yocto build output folder

The file rtlinux.bin will be the newly created RT-Linux binary image.

5.2.4 How to add an application to the default RT-Linux Image

In this example, we will show how to add the EC-Master version 3.1.4.11 demo application to the RT-Linux image. Here, the yocto packages is located in `~/yocto/rt_linux`.

Create a new folder `~/yocto/rt_linux/sources/yocto/recipes-core/ecmaster/files` and copy all EC-Master binaries to this folder.

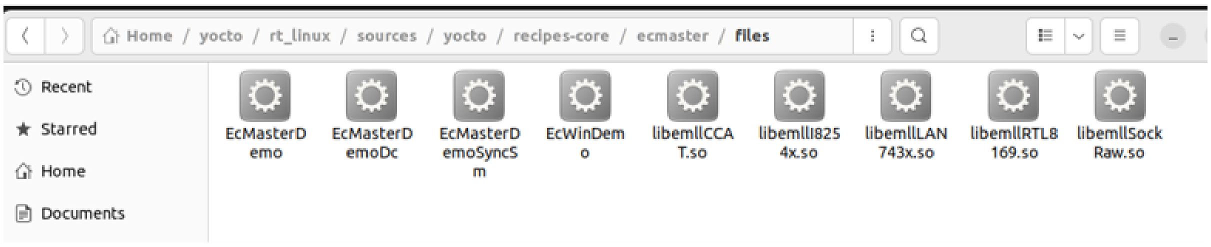


Fig. 5.4: Application files

Create a new Bitbake file ecmaster_3.1.4.11.bb at `~/yocto/rt_linux/sources/yocto/recipes-core/ecmaster`.

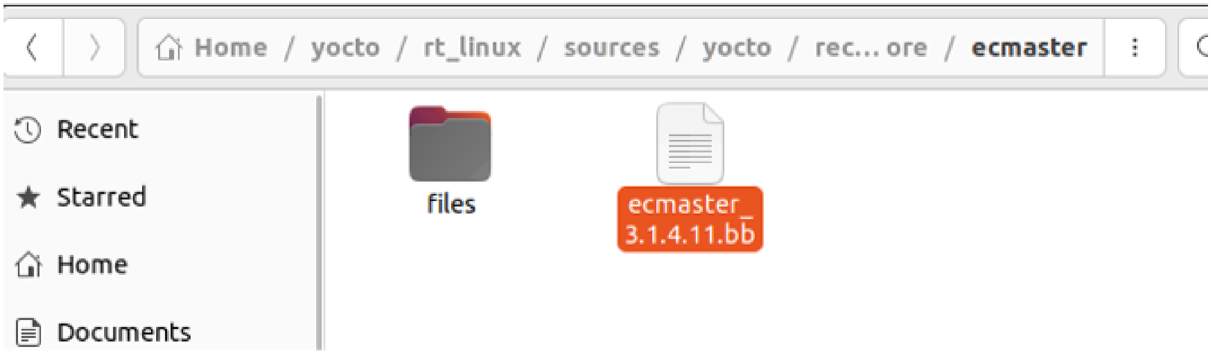

Home / yocto / rt_linux / sources / yocto / rec... ore / ecmaster
Recent
Starred
Home
Documents
files
ecmaster_3.1.4.11.bb

Fig. 5.5: Application Bitbake file

Add the following content into this file, this will become the build recipe which includes

- Header with a description, license information and dependencies
- List of source application files
- Installation instructions and package name (ecmaster)

```
SUMMARY = "EtherCAT Master"
LICENSE = "EC-Master_Standard-LicenseAgreement-V2.5.1"
LIC_FILES_CHKSUM = "file://${THISDIR}/../../../yocto_src/
↪licenses/EC-Master_Standard-LicenseAgreement-V2.5.1;
↪md5=7ba5f2117d3492ee30f5d68213d3324f"
DEPENDS = "librtos "
SRC_URI = "file://EcMasterDemo \
  file://libemllCCAT.so \
  file://libemllI8254x.so \
  file://libemllLAN743x.so \
  file://libemllRTL8169.so \
  file://libemllSockRaw.so \
  "

S = "${WORKDIR}„

do_install() {
  install -d ${D}/usr/bin/
  install -m 0755 ${WORKDIR}/EcMasterDemo ${D}/usr/bin/
  install -m 0755 ${WORKDIR}/libemllCCAT.so ${D}/usr/bin/
  install -m 0755 ${WORKDIR}/libemllI8254x.so ${D}/usr/bin/
  install -m 0755 ${WORKDIR}/libemllLAN743x.so ${D}/usr/bin/
  install -m 0755 ${WORKDIR}/libemllRTL8169.so ${D}/usr/bin/
  install -m 0755 ${WORKDIR}/libemllSockRaw.so ${D}/usr/bin/
}

FILES:${PN} += " \
/usr/bin/EcMasterDemo \
/usr/bin/libemllCCAT.so \
/usr/bin/libemllI8254x.so \
/usr/bin/libemllLAN743x.so \
/usr/bin/libemllRTL8169.so \
/usr/bin/libemllSockRaw.so"

# skip QA check because we deliver .so files inside non dev packet
```

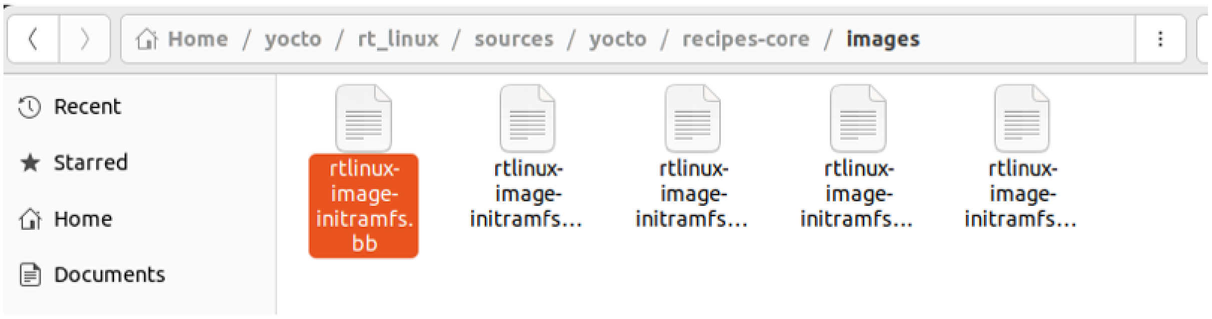
(continues on next page)

(continued from previous page)

```
INSANE_SKIP:${PN} += "ldflags"

PROVIDES += "ecmaster"
```

Adjust the .../`rtlinux-image-initramfs.bb` Bitbake file.



Home / yocto / rt_linux / sources / yocto / recipes-core / images
Recent
Starred
Home
Documents
rtlinux-image-initramfs.bb
rtlinux-image-initramfs...
rtlinux-image-initramfs...
rtlinux-image-initramfs...
rtlinux-image-initramfs...

Fig. 5.6: initramfs Bitbake file

add the `ecmaster` Package:

```
PACKAGE_INSTALL += " \
                   kernel-modules \
                   packagegroup-core-ssh-openssh \
                   openssh-sftp-server
                   gdbserver
                   glibc-utils
                   libstdc++ \
                   coreutils gdb tzdata tzdata-europe \
                   ecmaster \
                   "
```

Finally rebuild the image:

```
$ ./build.sh
```

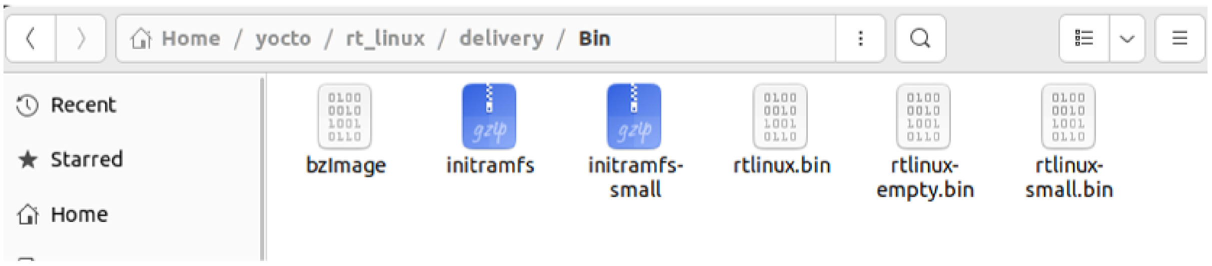


Fig. 5.7: Yocto build output folder

The file `rtlinux.bin` will be the newly created RT-Linux binary image.

5.3 Linux kernel real time configuration

RT-Linux is shipped with the kernel, optimized for real time usage. Customers can configure the kernel, if necessary, for their specific needs.

The Linux configuration is stored in the `file .config`. It can be changed with the `make menuconfig` command.

Here you can find the most important settings which are required to assure real-time behaviour.

`CONFIG_VMF` - Enables VMF paravirtualization code. Without this setting RT-Linux will be not able to start. Must be set.

`CONFIG_PREEMPT_RT_FULL` - Enables real time. Should be set.

`CONFIG_HIGH_RES_TIMERS` - Enables high resolution timers. Without these timers, applications will be not able to sleep less than a single OS tick. Its tick is controlled with the CONFIG_HZ setting.

Power management is controlled by Host and must be disabled in Linux: `CONFIG_PM`, `CONFIG_POWER_SUPPLY`, `CONFIG_ACPI`, `CONFIG_SUSPEND`, `CONFIG_LOCKUP_DETECTOR`.

CPU frequency changes, idle states and overheating control will have bad effects on the real-time behaviour. Thus, the following settings should not be set: `CONFIG_CPU_FREQ`, `CONFIG_CPU_IDLE`, `CONFIG_X86_MCE`.

The following devices are not supported by RT-Linux and should be disabled:

- USB devices: `CONFIG_USB_SUPPORT`
- Parallel: `CONFIG_PARPORT`
- Disk devices: `CONFIG_CDROM_PKTCDVD`, `CONFIG_SCSI`
- Graphics devices: `CONFIG_AGP`, `CONFIG_FB`
- HPET should be controlled by VMF: `CONFIG_HPET`

5.4 Security

The default RT-Linux image covers the needs of most customers.

In a typical use case, the Linux part of a RT-Linux based application does not have any connection to the Internet. In case the Linux part needs access to the outside network (and possibly is connected to the Internet also), for example by assigning a PCI network card to Linux, security measures may be required.

Here you will find some recommendations to increase security of the Linux part in such cases:

- The Telnet server should be disabled by setting the `notelnet` parameter to `yes`. You can build you own kernel image without the Telnet server.
- OpenBSD Secure Shell server may be disabled by setting the `nossh` parameter to `yes`. You can build you own kernel image without the SSH server.
- The default root password should be changed. It is defined at `Source/yocto/conf/layer.conf` file.
- Default certificates for SSH should be changed by changing the files at `Source/yocto/recipes-core/initrdscripts/files/etc/ssh` folder.

- For 64-bit Kernel the `nopti` parameter should be removed. You should set `cmd_line` parameter to `auto`

5.5 How to add LTTng to the RtLinux image

To include the different LTTng tools in your system, do the following:

- Add the following line to the `sources/yocto/recipes-core/images/rtlinux-image-initramfs.bb` file

```
PACKAGE_INSTALL += " lttng-tools lttng-modules lttng-ust "
```

- Add the line to the `sources/yocto/recipes-kernel/linux/linux-yocto-rt_5.15.bb` file:

```
SRC_URI:append = " file://trace.cfg "
```

- Add the line to the `sources/yocto/recipes-kernel/linux/linux-yocto-rt/trace.cfg` file

```
CONFIG_KPROBES=y
```

- Build the image and copy rtlinux.bin to the host machine.

To test LTTng you can follow “LTTng” chapter from the “Yocto Project Profiling and Tracing Manual”. Simple test could be done with following command in RtLinux guest:

```
lttng create
```