rexroth
A Bosch Company

Application Manual

Container Engine App

Using Docker® Images on ctrlX OS 03VRS

R911425234 Edition 01

Disclaimer

The data specified above only serve to describe the product. As our products are constantly being further developed, no statements concerning a certain condition or suitability for a certain application can be derived from our information. The information given does not release the user from the obligation of own judgment and verification. It must be remembered that our products are subject to a natural process of wear and aging.

Table of contents

1 About this documentation

Editions of this documentation

Edition	Release date	Note
01	2024-12	First edition Container Engine App Version DOE-V-0302

2 Important directions on use

2.1 Intended use

2.1.1 Introduction

Rexroth products are developed and manufactured to the state-of-the-art.

The products are tested prior to delivery to ensure operational safety and reliability.

⚠ WARNING	**Personal injury and damage to property due to incorrect use of products!** The products may only be used as intended. Failure to use the products as intended may cause situations resulting in property damage and personal injury.
NOTICE	**Damages resulting from unintended use** Rexroth As the manufacturer does not assume any warranty, liability or compensatory claims for damages resulting from unintended use of the products. The user alone shall bear the risks of an unintended use of the products. Before using Rexroth products, make sure that all the prerequisites for an intended use of the products are met: – Personnel that in any way, shape or form uses Rexroth products must first read and understand the relevant safety instructions and be familiar with their intended use – Leave hardware products in their original state, i.e., do not make any structural modifications. It its not permitted to decompile software products or alter source codes – Do not install damaged or defective products or commission them – It has to be ensured that the products have been installed as described in the relevant documentation

2.1.2 Areas of use and application

Products of the ctrlX series are suitable for Motion/Logic applications.

NOTICE	Products of the ctrlX series may only be used with the accessories, mounting parts, and other components specified in this documentation. Components that are not expressly mentioned must neither be attached nor connected. The same applies to cables and lines. Only to be operated with the hardware component configurations and combinations expressly specified and with the software and firmware specified in the corresponding documentations and functional descriptions.

Products of the ctrlX series are suitable for single-axis as well as for multi-axis drive and control tasks. Device types with different equipment and interfaces are available for using the system in specific applications.

Typical areas of application:

- Building automation
- IoT and Security Gateway or Device
- Handling & Robotic

Controls of the ctrlX CORE series may only be operated under the mounting and installation conditions, in the position of normal use and under the ambient conditions (temperature, degree of protection, humidity, EMC, etc.) specified in the related documentations.

2.2 Unintended use

"Unintended use" refers to using the ctrlX products outside of the above-mentioned areas of application or under operating conditions and technical data other than described and specified in the documentation.

ctrlX products must not be used if they are exposed to following conditions:

- Operating conditions that do not meet the specified ambient conditions. Operation under water, under extreme temperature fluctuations or under extreme maximum temperatures is prohibited
- Applications that have not been expressly authorized by Rexroth

3 Safety instructions

The Safety instructions contained in the available application documentation feature specific signal words (DANGER, WARNING, CAUTION or NOTICE) and, where required, a safety alert symbol (in accordance with ANSI Z535.6-2006).

The signal word is meant to draw the reader's attention to the safety instruction and identifies the hazard severity.

The safety alert symbol (a triangle with an exclamation point), which precedes the signal words DANGER, WARNING and CAUTION, is used to alert the reader to personal injury hazards.

The Safety instructions in this documentation are designed as follows:

⚠ DANGER	In case of non-compliance with this safety instruction, death or serious injury **will** occur.
⚠ WARNING	In case of non-compliance with this safety instruction, death or serious injury **could** occur.
⚠ CAUTION	In case of non-compliance with this safety instruction, minor or moderate injury could occur.
NOTICE	In case of non-compliance with this safety instruction, property damage could occur.

4 Introduction and overview

Definitions of terms

The following documentation uses terms which are described in the ➦Glossary.

Useful web links

➦ctrlX Store on the web
➦How-to range
➦ctrlX AUTOMATION FORUM
➦ctrlX AUTOMATION Community

4.1 Container Engine app – Basics

4.1.1 Functional principle

The Container Engine app provides a Docker engine on ctrlX devices, which enables the execution of Docker images.

Multiple Docker images can be installed and operated on the ctrlX device via separate Docker image apps.

Docker images are configured and started via the Docker container configuration file **docker-compose.yml**, see: ➦Example

Snap templates for ctrlX devices

A snap template can be used to create a Docker image application for ctrlX devices from a Docker image and the corresponding Docker configuration files.

4.1.2 Installation and integration into the web interface of the ctrlX device

The Container Engine app can be transferred and installed on a ctrlX device either via the ctrlX Store or via the ctrlX Device Portal, see:

➦ctrlX Store on the web
➦Web documentation for the ctrlX Device Portal

The procedure for installing apps on ctrlX devices is described in the documentation for ctrlX Runtime, see: ➦Web documentation

By installing the Container Engine app, the web interface of the ctrlX device is extended by the following elements:

➦Window – “Images”
➦Window – “Containers”

4.1.3 Licensing

The operation of the Container Engine app on a ctrlX device is subject to licensing and requires the following license:

Type code	Material number
SWL-XC*-DOE-DOCKERENGINE*-NNNN	R911409943

4.1.4 Users and authorizations

The user authentication of the Container Engine app is linked to the user administration of the ctrlX device, see: ➦Web documentation

For the configuration and operation of the Container Engine app, the following authorizations have to be available:

- Managing the Container Engine configuration
- Starting/stopping the Container and viewing the Container Engine configuration
- View Container Engine configuration

The authorizations can be managed and configured in the web interface of the ctrlX device, see: ➦Web documentation

Insufficient authorizations may mean that no data can be displayed or that buttons have no function.

4.2 Interfaces

The Container Engine app contains a Docker engine with Docker daemon (dockerd) and the Docker command line interface (Docker CLI).

Furthermore, the app provides two content interfaces used to exchange data between the Container Engine app and a Container Image app.

The following UML diagram shows the interfaces and the data flow schematically:

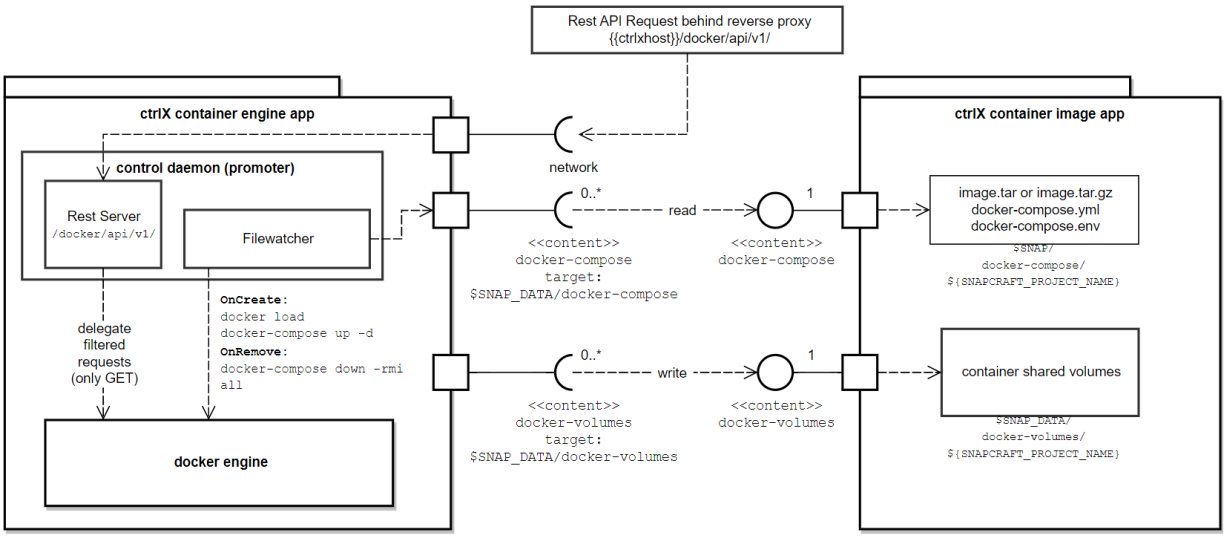

Fig. 1: Interface and data flow

4.2.1 "docker-compose" content interface

The Docker Compose interface is used to provide the Docker images and the associated configuration files for the Container Engine app.

The following files are provided via Docker-Compose:

```
└── docker-compose
    ├── docker-compose.env              ← Docker-Compose
Variablen (optional)
    ├── docker-compose.yml              ← Docker-Container
Konfiguration
    ├── image-1.tar / image-1.tar.gz
    ├── …                              ← Docker-Image Archiv(e)
    └── image-n.tar / image-n.tar.gz
```

4.2.2 "docker-volumes" content interface

The Docker Volumes content interface allows write access from a Docker container to an image app directory. A Docker container can write from the ctrlX Container Engine app via this interface to a writable directory of the Container Image app.

Example directory:

```
$SNAP_DATA/docker-volumes/{snap-name}
```

Exemplary excerpt from a docker-compose.yml for the assignment of a Docker volume with write access:

```
services:
  {service}:
    image: {image}
    volumes:
      - ${SNAP_DATA}/docker-volumes/{snap-name}/data:/data:rw
```

4.2.3 Docker engine API

The REST interface of the Docker Engine API can be accessed via the following address:

https://{host}/docker/api/v1

A complete description of the Docker REST API can be found under the following web link:

➦ https://docs.docker.com/engine/api/latest/

Example of a REST command via curl - here: retrieving the existing Docker images as json:

```
curl --location --request GET "https://{host}/docker/api/v1/images/json" --header "Authorization: {token}"
```

5 Creating a Container Image app

Bosch Rexroth does not provide any Docker files and Docker images for commercial use.

The creation and distribution of Docker images with the ctrlX Container app is the responsibility of the user.

The following instructions describe the technical steps for creating an image.

In any case, make sure that the license conditions are complied with when using and distributing Docker images, in particular for free software and open source software (FOSS).

5.1 Requirements

To create a ctrlX Container Image app, the user requires the appropriate tools and has to familiar with Docker and Snapcraft.

The Linux Ubuntu 22.04 LTS operating system is recommended as development platform.

Ubuntu 22.04 LTS can be run in a virtual machine for development purposes. However, the virtualization should allow the use of snapd, e.g. to be able to obtain Snapcraft or Docker with the latest releases via the Snapcraft Store. In the current version of the Windows subsystem for Linux (WSL), snapd cannot be operated.

5.1.1 Docker

To create a container image app, a Docker image (image.tar or image.tar.gz) has to be available.

The installation of a Docker engine on Ubuntu is described under the following web link:

➦https://docs.docker.com/engine/install/ubuntu/

It is recommended to use a release of the Docker engine that supports the creation of multi-platform images. For example, the latest Docker engine can be installed via the Snapcraft store, see:

➦https://snapcraft.io/docker → sudo snap install docker

To be able to run Docker with the authorizations of a standard user, the following command sequence has to be entered:

```
sudo addgroup --system docker
sudo adduser $USER docker
newgrp docker
sudo snap disable docker
sudo snap enable docker
```

Also refer to:

➦ https://github.com/docker-snap/docker-snap/blob/main/README.md Snapcraft

5.1.2 Snapcraft

The installation of Snapcraft can be executed according to these instructions:

➦https://snapcraft.io/install/snapcraft/ubuntu

To use the latest release of Snapcraft on Ubuntu 22.04, Snapcraft should be installed via the Snapcraft Store:

➦https://snapcraft.io/install/snapcraft/ubuntu

```
sudo snap install snapcraft --classic
```

5.2 Creating a Docker image

First, a Docker image has to be created and loaded into the Docker engine.

A Docker image can be created via the Docker Command Line Interface (Docker CLI). The corresponding Docker documentation can be used, see:

➦https://docs.docker.com/reference/

An existing Docker image can be downloaded via ➦Docker Hub.

In the following example, the Docker image "eclipse-mosquitto" for the target platform arm64 is saved in a Docker image file via a shell script using Docker CLI commands:

Example

```
IMAGE_NAME="eclipse-mosquitto"
IMAGE_TAG="latest"
TARGET_ARCH=arm64
docker pull ${IMAGE_NAME}:${IMAGE_TAG} --platform ${TARGET_ARCH}
docker save ${IMAGE_NAME}:${IMAGE_TAG} | gzip > ./docker-compose/image.tar.gz
docker rmi ${IMAGE_NAME}:${IMAGE_TAG}
```

5.3 Snap configuration

The following interfaces are already set in the snapcraft.yaml in the ➦ snap template example. Only a few adjustments have to be made.

An example Snapcraft configuration of the Mosquitto Image app can be found here: ➦Snap templates

5.4 Docker Compose variables

Environment variables can be created in the docker-compose.env file. These variables can be accessed in docker-compose.yml.

Example of a docker-compose.env:

```
IMAGE_NAME=eclipse-mosquitto
IMAGE_TAG=latest
```

You can find documentation on the topic of Docker Compose variables under the following web link:

➦ https://docs.docker.com/compose/environment-variables/

5.5 Docker Compose configuration

The Docker application is configured via the docker-compose.yml file.

You can find instructions for the Docker Compose configuration here:

➦ https://docs.docker.com/compose/

Example of a docker-compose.yml for eclipse-mosquitto, see:

➦ "eclipse-mosquitto” template

In the example, the Docker volumes are assigned with write access to a directory within the image app.

The following directories can be accessed from the Docker container via the assignment in the Docker Compose configuration with write permissions:

```
${SNAP_DATA}/docker-volumes/mosquitto-docker/data
${SNAP_DATA}/docker-volumes/mosquitto-docker/log
```

The "restart: on-failure" option should always be set so that the Docker containers can also be started in the event of an error.

5.6 Creating the Image app

To create the snap with the Docker content, the following snapcraft commands have to be executed in the console:

Clear Snapcraft build directories

```
snapcraft clean
```

Create snap with corresponding target architecture:

For target architecture arm64:

```
snapcraft --target-arch=arm64
```

For target architecture amd64:

```
snapcraft --target-arch=amd64
```

6 Diagnostics

6.1 SSH console

Currently, the Container Engine and Container Image apps can only be diagnosed to a limited extent via the ctrlX web interface.

The use of an SSH console with root rights is recommended for a comprehensive diagnostics (see ➦ Further information on page 12).

An SSH console can be opened under Windows with the following command:

```
ssh {user}@{host}
```

Example:

```
ssh boschrexroth@192.168.1.1
```

6.2 Container Diagnosis log

The Diagnosis log of the installed containers are displayed in the web interface in the Containers window, see: ➦Window – “Containers”

Call path:

“Page navigation → Container Engine → Containers” Click on ⓘ in the column of the relevant container → Tab Diagnosis log

6.3 Docker command line interface

The Docker command line interface can be used to retrieve information from the Docker Engine and to execute commands.

A complete documentation is available under the following link:

➦https://docs.docker.com/engine/reference/commandline/cli/

Examples of important commands for the of Docker image diagnostics

```
sudo ctrlx-docker.docker images
sudo ctrlx-docker.docker ps -a
sudo ctrlX-docker.docker logs {{container-id}}
```

7 Troubleshooting tips

This section deals with potential sources of error that can occur when creating a Container Image app.

7.1 IP forwarding

If an HTTP server is to be accessed in the Docker container, the option “Enable IP forwarding” has to be enabled in the network settings.

The setting can be configured in the web interface of the ctrlX device, see: ➦Related web documentation

7.2 Target architecture

When creating a Docker image file, the correct architecture for the target system has to be specified.

If the Docker image is loaded via a Docker repository such as Docker Hub, the target architecture can be transferred using the "platform" switch.

Example

```
docker pull ${IMAGE_NAME}:${IMAGE_TAG} --platform ${TARGET_ARCH}
```

Target architecture for arm64 systems, e.g. ctrlX CORE X3

```
docker pull eclipse-mosquitto:latest --platform arm64
```

Target architecture for arm64 systems, e.g. for ctrlX CORE Virtual

```
docker pull eclipse-mosquitto:latest --platform amd64
```

7.3 Storage space

When selecting a Docker image in Docker Hub, check the image size as storage space is limited, depending on the ctrlX CORE device type.

For example, the size of the Docker image should not exceed 300 MB for a ctrlX device type X3.

For Linux-based Docker images, the Linux distribution "Alpine" is recommended to minimize the image size, see here:
➦ https://hub.docker.com/_/alpine/tags

7.4 Behavior after control restart

To ensure that the running containers in the Docker Engine are restarted automatically in the event of a ctrlX device restart, the option `restart` has to be set to `always` in the "docker-compose.yml" configuration file, refer to:

- ➦ Docker-compose
- ➦ https://docs.docker.com/compose/compose-file/compose-file-v3/#restart

8 Snap templates

8.1 Snap template shell scripts

ctrlX Container Image snaps are created using shell scripts in the following steps.

8.1.1 Build Docker content

The content of the docker-compose directory is supplemented or completed with the help of the build_content.sh shell script:

1. Create Docker image archives 'image.tar'
2. Create Docker variable file 'docker-compose.env'

8.1.2 Build Snap

With the help of the build_snap.sh shell script, the snap build is first prepared and then created for the corresponding target platform:

1. Delete the existing snap
2. Delete the existning snapcraft configuration
3. Create snap again for a specific target architecture

8.1.3 Build All

By using the Shell script build_all.sh, the steps ➦ Snap template shell scripts and ➦ Snap template shell scripts are executed in the correct order and for the corresponding target platform.

8.2 Template "hello-web"

The "hello-web" template demonstrates the creation of a simple Docker image via a Docker file.

8.2.1 File structure

```
.
├── build_all.sh
├── build_content.sh
├── build_snap.sh
├── configs
│   └── package-assets
│       └── hello-web.package-manifest.json
├── docker
│   ├── amd64
│   │   ├── Dockerfile
```

```
│   │   └── web.sh
│   └── arm64
│       ├── Dockerfile
│       └── web.sh
├── docker-compose
│   └── docker-compose.yml
└── snap
    └── snapcraft.yaml
```

8.2.2 Package assets

The configs directory is provided including the package-manifest.json file with the content interface package assets. The package assets configuration is documented here: ➦ https://boschrexroth.github.io/ctrlx-automation-sdk/package-assets.html

Example: hello-web.package-manifest.json

```
{
  "$schema": "https://json-schema.boschrexroth.com/ctrlx-automation/ctrlx-core/apps/package-manifest/package-manifest.v1.1.schema.json",
  "version": "1.0.0",
  "id": "hello-web",
  "menus": {
    "sidebar": [
      {
        "id": "hello-web.dashboard",
        "title": "Hello Web",
        "icon": "bosch-ic-worldwideweb",
        "target": "_blank",
        "link": "http://${hostname}:8188",
        "permissions": []
      }
    ],
    "overview": [
      {
        "id": "hello-web.dashboard",
        "title": "Hello Web",
        "icon": "bosch-ic-worldwideweb",
        "target": "_blank",
        "link": "http://${hostname}:8188",
        "permissions": []
      }
    ]
  }
}
```

8.2.3 Docker files

Docker file amd64

```
FROM alpine:latest
LABEL maintainer="support@boschrexroth.com"
LABEL description="hello-web"
COPY ./web.sh /web/
WORKDIR /web
EXPOSE 8188
ENTRYPOINT [ "./web.sh" ]
```

Docker file arm64

```
FROM arm64v8/alpine:latest
LABEL maintainer="support@boschrexroth.com"
LABEL description="hello-web"
COPY ./web.sh /web/
WORKDIR /web
EXPOSE 8188
ENTRYPOINT [ "./web.sh" ]
```

Shell script web.sh

```
#!/bin/sh
while true; do
  (echo -e "HTTP/1.1 200 OK\r\n" ; echo -e "\n\tMy website has date function" ; echo -e "\t$(date)\n") | nc -l
-p 8188
done
```

8.2.4 Snapcraft

snapcraft.yaml

```
name: docker-hello-web
version: '2.2.0'
base: core22
summary: Docker example with simple web server based on netcat (nc)
description: |
  This snap contains a docker image with a simple web server.
  The files 'image.tar', 'docker-compose.yml' and 'docker-compose.env'
  are provided via content-interface 'docker-compose'.
  The content-interface 'docker-volumes' provides the container
  access to a directory inside this snap with write permissions.

grade: stable
confinement: strict

parts:
  docker-compose:
    plugin: dump
    source: ./docker-compose
    organize:
      '*': docker-compose/${SNAPCRAFT_PROJECT_NAME}/
  configs:
    source: ./configs
    plugin: dump
    organize:
      'package-assets/*': package-assets/${SNAPCRAFT_PROJECT_NAME}/

slots:
  docker-compose:
    interface: content
    content: docker-compose
    source:
      read:
        - $SNAP/docker-compose/${SNAPCRAFT_PROJECT_NAME}
  docker-volumes:
    interface: content
    content: docker-volumes
    source:
      write:
        - $SNAP_DATA/docker-volumes/${SNAPCRAFT_PROJECT_NAME}
  package-assets:
    interface: content
    content: package-assets
    source:
      read:
        - $SNAP/package-assets/${SNAPCRAFT_PROJECT_NAME}
  package-run:
    interface: content
    content: package-run
    source:
      write:
      - $SNAP_DATA/package-run/${SNAPCRAFT_PROJECT_NAME}
```

8.2.5 Docker-compose

docker-compose.yml

```
version: '3.7'
services:
  brc-web:
    container_name: "hello-web"
    image: ${IMAGE_NAME}:${IMAGE_TAG}
    ports:
      - "8188:8188"
    stdin_open: false
    tty: false
    restart: always
    volumes:
      - brc-web-data:/data
volumes:
  brc-web-data:
```

8.2.6 Shell scripts

build_content.sh

```
#!/bin/bash
TARGET_ARCH=$(dpkg --print-architecture)
if [[ -n $1 ]]; then
    TARGET_ARCH=$1
fi
echo TARGET_ARCH: ${TARGET_ARCH}
DOCKER_CLI=/snap/bin/docker
IMAGE_NAME=hello-web
IMAGE_TAG='latest'
echo --- create docker image with platform ${TARGET_ARCH}
rm -f -v ./docker-compose/*.tar
${DOCKER_CLI} build -t ${IMAGE_NAME}:${IMAGE_TAG} ./docker/${TARGET_ARCH}
${DOCKER_CLI} save ${IMAGE_NAME}:${IMAGE_TAG} | gzip > ./docker-compose/image.tar.gz
${DOCKER_CLI} image rm ${IMAGE_NAME}:${IMAGE_TAG}
echo --- create docker-compose environment file
rm -f ./docker-compose/docker-compose.env
echo IMAGE_NAME=${IMAGE_NAME} >> ./docker-compose/docker-compose.env
echo IMAGE_TAG=${IMAGE_TAG} >> ./docker-compose/docker-compose.env
```

build_snap.sh

```
#!/bin/bash
TARGET_ARCH=$(dpkg --print-architecture)
if [[ -n $1 ]]; then
    TARGET_ARCH=$1
fi
echo TARGET_ARCH: ${TARGET_ARCH}
echo --- clean snap
snapcraft clean --destructive-mode
echo --- build snap with TARGET_ARCH ${TARGET_ARCH}
snapcraft --destructive-mode --enable-experimental-target-arch --target-arch=${TARGET_ARCH}
```

build_all.sh

```
#!/bin/bash
TARGET_ARCH=$(dpkg --print-architecture)
if [[ -n $1 ]]; then
    TARGET_ARCH=$1
fi
echo TARGET_ARCH: ${TARGET_ARCH}
echo --- build content
bash build_content.sh ${TARGET_ARCH}
echo --- build snap
bash build_snap.sh ${TARGET_ARCH}
```

8.3 "eclipse-mosquitto” template

The "eclipse-mosquitto" template demonstrates the use of an existing Docker image in the Docker hub (➥ https://hub.docker.com/).

File structure

```
.
├── build_all.sh
├── build_content.sh
├── build_snap.sh
├── docker-compose
│   ├── config
│   │   └── mosquitto.conf
│   └── docker-compose.yml
└── snap
    └── snapcraft.yaml
```

8.3.1 Snapcraft

snapcraft.yaml

```
name: docker-mosquitto
version: '2.2.0'
base: core22
summary: Docker example with 'eclipse-mosquitto' image
description: |
  This snap contains the docker image 'eclipse-mosquitto:latest'.
  The files 'image.tar', 'docker-compose.yml' and 'docker-compose.env'
  are provided via content-interface 'docker-compose'.
  The content-interface 'docker-volumes' provides the container
  access to a directory inside this snap with write permissions.

grade: stable
confinement: strict

parts:
  docker-compose:
    plugin: dump
    source: ./docker-compose
    organize:
      '*': docker-compose/${SNAPCRAFT_PROJECT_NAME}/
slots:
  docker-compose:
    interface: content
    content: docker-compose
    source:
      read:
        - $SNAP/docker-compose/${SNAPCRAFT_PROJECT_NAME}
  docker-volumes:
    interface: content
    content: docker-volumes
    source:
      write:
        - $SNAP_DATA/docker-volumes/${SNAPCRAFT_PROJECT_NAME}
```

8.3.2 Docker-compose

config

The config directory, including the mosquitto.conf file, is provided with the Docker-Compose content interface. This makes it possible to access the mosquitto.conf file in the Docker container via the volume assignment in docker-compose.yml.

docker-compose.yml

```
version: "3.7"
services:
  mosquitto:
    image: ${IMAGE_NAME}:${IMAGE_TAG}
    container_name: mosquitto
    ports:
      - 1883:1883
      - 9001:9001
    volumes:
      - ${SNAP_DATA}/docker-compose/docker-mosquitto/config:/mosquitto/config
      - ${SNAP_DATA}/docker-volumes/docker-mosquitto/data:/mosquitto/data
      - ${SNAP_DATA}/docker-volumes/docker-mosquitto/log:/mosquitto/log
    restart: on-failure
```

8.3.3 Shell scripts

build_content.sh

```
#!/bin/bash
TARGET_ARCH=$(dpkg --print-architecture)
if [[ -n $1 ]]; then
    TARGET_ARCH=$1
fi
echo TARGET_ARCH: ${TARGET_ARCH}
IMAGE_NAME="eclipse-mosquitto"
IMAGE_TAG="latest"
DOCKER_CLI="/snap/bin/docker"
echo --- create ./docker-compose/docker-compose.env
rm -v -f ./docker-compose/docker-compose.env
echo IMAGE_NAME=${IMAGE_NAME} >> ./docker-compose/docker-compose.env
echo IMAGE_TAG=${IMAGE_TAG} >> ./docker-compose/docker-compose.env
echo --- create docker image with platform ${TARGET_ARCH}
rm -f -v ./docker-compose/*.tar
${DOCKER_CLI} pull ${IMAGE_NAME}:${IMAGE_TAG} --platform ${TARGET_ARCH}
${DOCKER_CLI} save ${IMAGE_NAME}:${IMAGE_TAG} | gzip > ./docker-compose/image.tar.gz
${DOCKER_CLI} rmi ${IMAGE_NAME}:${IMAGE_TAG}
```

build_snap.sh

```
#!/bin/bash
TARGET_ARCH=$(dpkg --print-architecture)
if [[ -n $1 ]]; then
    TARGET_ARCH=$1
fi
echo TARGET_ARCH: ${TARGET_ARCH}
echo --- clean snap
snapcraft clean --destructive-mode
echo --- build snap with architecture ${TARGET_ARCH}
snapcraft --destructive-mode --enable-experimental-target-arch --target-arch=${TARGET_ARCH}
```

build_all.sh

```
#!/bin/bash
TARGET_ARCH=$(dpkg --print-architecture)
if [[ -n $1 ]]; then
    TARGET_ARCH=$1
fi
echo TARGET_ARCH: ${TARGET_ARCH}
echo --- build content
bash build_content.sh ${TARGET_ARCH}
echo --- build snap
bash build_snap.sh ${TARGET_ARCH}
```

9 ctrlX UI – Elements

9.1 Windows

9.1.1 Window – “Images”

The window “Images” displays the list of Docker images installed in the Container Engine.

For each image, several properties, such as name and size are displayed. Via “Details”, all properties of an image can be displayed.

Related topics:
➦ Information about the Container Engine and the Container Engine app

Call:

ctrlX OS side navigation *“Container Engine → Images”*

Elements of the “Images” window

Interface element	Description
Table	“Name” Image name
	“Tags” Image tag See ➦ link to the web documentation
	“Maintainer” Image publisher
	“Size” Image name
	“Created” Time of image creation
	“Details” ⓘ Click on the interface to open the “Further information” window of the image

Additional information

- ➦ Chapter 4 Introduction and overview on page 7
- ➦ Chapter 9.1.2 Window – “Containers” on page 19

9.1.2 Window – “Containers”

The window “Containers” displays the list of containers in the Container Engine and their current state

Related topics:
➦ Information about the Container Engine and the Container Engine app

Call:

ctrlX OS side navigation *“Container Engine → Containers”*

Elements of the “Containers” **window**

Interface element	Description
Table	“Name” Container name
	“Version” Container version
	“Created” Time of container creation
	“IP adress” IP address of the container
	“Ports” Container ports
	“Image” The image is displayed Click on the image name to open the “Images” window to display “Further information”
	“State” Container status
	“Actions” Contains more interfaces ▷ “Start” Starting the container □ “Stop” Stopping the container ↻ “Restart” Container restart ⏸ “Pause” Pausing the container ✕ “Kill” Deleting the container
	“Details” ⓘ Clicking on the button opens a window with further information on the container with the “Diagnosis log” and “Further information” tabs. Use ↻ to refresh the view in the “Diagnosis log” tab

Additional information

- ➦Chapter 4 Introduction and overview on page 7
- ➦Chapter 9.1.1 Window – “Images” on page 19

10 Further documentation

10.1 Overview

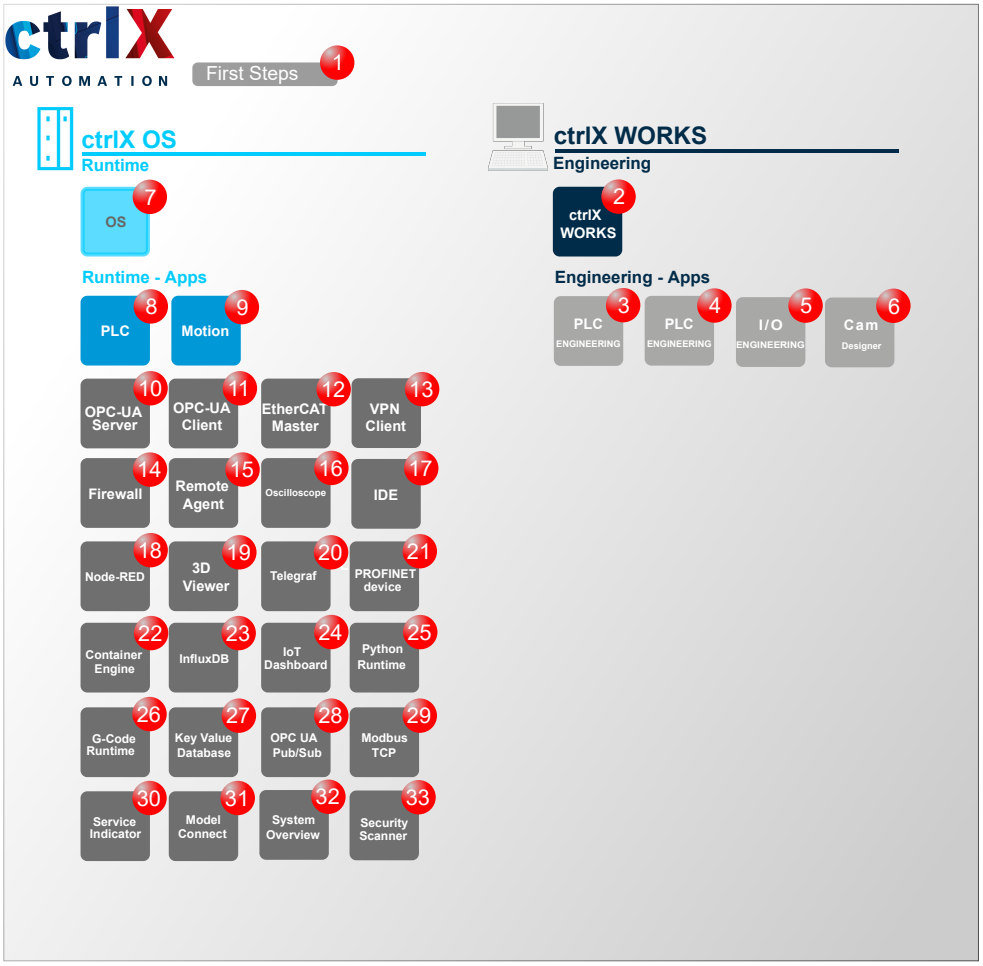

Fig. 2: Overview of further documentation

10.2 ctrlX AUTOMATION

No.	Documentation
1	**ctrlX WORKS - First Steps** Quick Start Guide ➦ Link to the web documentation Ordering information: • DOK-XWORKS-F*STEP*****-QU01-EN-P • R911403760

10.3 ctrlX WORKS

No.	Documentation
2	**ctrlX WORKS - Basic System 03VRS** Application manual ➦ Link to the web documentation Ordering information: ● DOK-XWORKS-WRK***V03**-APRS-EN-P ● R911423376
3	**ctrlX PLC Engineering - PLC Programming System 03VRS** Application manual ➦ Link to the web documentation Ordering information: ● DOK-XPLC**-PLE***V03**-APRS-EN-P ● R911423378
4	**ctrlX PLC Engineering - PLC Libraries 03VRS** Reference ➦ Link to the web documentation Ordering information: ● DOK-XPLC**-LIB***V03**-RERS-EN-P ● R911423456
5	**ctrlX I/O Engineering - Field Bus Configuration for ctrlX OS 03VRS** Application manual ➦ Link to the web documentation Ordering information: ● DOK-XIO***-IOE***V03**-APRS-EN-P ● R911423380
6	**ctrlX Cam Designer - Configuring ctrlX MOTION Cams 03VRS** Application manual ➦ Link to the web documentation Ordering information: ● DOK-XWORKS-CAM***V03**-APRS-EN-P ● R911427217

10.4 ctrlX OS

No.	Documentation
7	**ctrlX OS - Operating System for ctrlX CORE Control Devices 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-XCR***V03**-APRS-EN-P • R911423382
	ctrlX OS - Data Layer Node 03VRS Reference ➦Link to the web documentation Ordering information: • DOK-XCORE*-DL****V03**-RERS-EN-P • R911423384
	ctrlX OS - Diagnostics 03VRS Reference ➦Link to the web documentation Ordering information: • DOK-XCORE*-DIAG**V03**-RERS-EN-P • R911423386

10.5 ctrlX OS Apps

No.	Documentation
8	**PLC App - PLC Runtime Environment for ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-PLC***V03**-APRS-EN-P • R911423401
9	**Motion App - Motion Runtime Environment for ctrlX CORE 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-MOT***V03**-APRS-EN-P • R911423405
10	**OPC UA Server App - OPC UA Server for ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-UAS***V03**-APRS-EN-P • R911423392

No.	Documentation
11	**OPC UA Client App - OPC UA Client for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-UAC***V03**-APRS-EN-P • R911423390
12	**EtherCAT Master App - EtherCAT Master for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-ECM***V03**-APRS-EN-P • R911423394
13	**VPN Client App - Remote Maintenance Software for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-VPN***V03**-APRS-EN-P • R911423388
14	**Firewall App - Security Functions for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-FRW***V03**-APRS-EN-P • R911423397
15	**Remote Agent App - ctrlX Device Portal-Connection for ctrlX OS Devices 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-RMA***V03**-APRS-EN-P • R911423399
16	**Oscilloscope App - Oscilloscope Function for ctrlX OS devices 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-OSC***V03**-APRS-EN-P • R911423407
17	**IDE App - Integrated Development Environment 02VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-IEN***V02**-APRS-EN-P • R911421612

No.	Documentation
18	**Node-RED App - Graphical Programming for ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-RED***V03**-APRS-EN-P • R911423403
19	**3D Viewer App - Browser-based 3D Kinematics Simulation for ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-3DV***V03**-APRS-EN-P • R911423411
20	**Telegraf App - Server Agent for Collecting Data in the Data Layer 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-TSA***V03**-APRS-EN-P • R911425238
21	**PROFINET Device App - PROFINET Device for ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-PND***V03**-APRS-EN-P • R911425232
22	**Container Engine App - Using Docker® Images on ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-DOE***V03**-APRS-EN-P • R911425234
23	**InfluxDB App - Influx Database Connection for ctrlX OS 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-IDB***V03**-APRS-EN-P • R911425240
24	**IoT Dashboard App - Data Visualization in Dynamic, Interactive Dashboards 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-GDB***V03**-APRS-EN-P • R911425248

No.	Documentation
25	**Python Runtime App - Python Runtime Environment for ctrlX CORE 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-PYR***V03**-APRS-EN-P • R911425244
26	**G-Code Runtime App - G-Code Interpreter for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-GCO***V03**-APRS-EN-P • R911425246
27	**Key Value Database App - Managing Data in the Data Layer 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-KVD***V03**-APRS-EN-P • R911425250
28	**OPC UA Pub/Sub App - OPC UA Pub/Sub for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-UAP***V03**-APRS-EN-P • R911423409
29	**Modbus TCP App - Modbus TCP Communication for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-MBT***V03**-APRS-EN-P • R911425236
30	**Service Indicator App -Service Indicator for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-SIN***V03**-APRS-EN-P • R911425242
31	**Model Connect App - Target for Model-Based Development and Simulation for ctrlX OS 03VRS** Application manual ➥ Link to the web documentation Ordering information: • DOK-XCORE*-MOC***V03**-APRS-EN-P • R911425252

No.	Documentation
32	**System Overview App - System Topology and System Information 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-SOV***V03**-APRS-EN-P • R911425254
33	**Security Scanner App -Inventory of Components 03VRS** Application manual ➦Link to the web documentation Ordering information: • DOK-XCORE*-SSC***V03**-APRS-EN-P • R911427698

11 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts provide you with advice and assistance. You can contact us **24/7**.

Service Germany

Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	**+49 9352 40 5060**
Fax:	**+49 9352 18 4941**
Email:	➦service.svc@boschrexroth.de
Internet:	➦http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide

Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information

To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

12 Glossary

12.1 Docker

A comprehensive and detailed glossary of Docker can be found here:
➦Link to the web documentation

Image

A Docker image is a memory image of a container. An image consists of several layers, which are read-only and cannot be changed. Several containers can be started from one image.

Container

A container is the runtime instance of an image.

A Docker container consists of:

- Docker image
- Execution environment
- Set of instructions

See ➦link to the web documentation

Volumes

Docker volumes provide a file system on the host system with or without write access that can be used by the container. See ➦ link to the web documentation

Engine

The Docker engine acts as client/server application and contains the following:

- A server with a daemon process dockerd
- Interfaces via which programs can communicate with the Docker daemon and instruct it
- A command line interface (CLI)

See ➦ link to the web documentation

Compose

Compose is a tool for configuring and executing complex applications with Docker. Compose is used to define a multi-container application in a single file. This application can be started with a single command using the compose file (docker-compose.yml). See ➦ link to the web documentation

12.2 Snapcraft

A comprehensive and detailed glossary of Snap and Snapcraft can be found here:

➦ Link to the web documentation

Snapcraft

Snapcraft is a command line tool for creating snaps and enables the creation of snaps and thus apps for ctrlX CORE. See ➦ link to the web documentation

Content interface

The Snapcraft content interface enables code and data to be shared from a producer snap to a consumer snap. See ➦ link to the web documentation

13 Index

Bosch Rexroth AG
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr a.Main
Germany
Tel. +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics

R911425234 01