rexroth
A Bosch Company

Application Manual

Container Engine App

Using Docker® Images on ctrlX CORE 01VRS

R911417855, Edition 01

Liability

The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.

DOK-XCORE*-DOCKER**V01-AP01-EN-P DC-AE/EPI5 (MiSc/PiaSt)

c2e7b15c673acc930a347e885d054cbf, 1, en_US

Table of contents

1 About this documentation 5

2 Important directions on use 7

2.1 Intended use. 7

2.1.1 Introduction. 7

2.1.2 Areas of use and application 7

2.2 Unintended use. 8

3 Safety instructions 9

4 Introduction and overview 11

4.1 Container Engine app - Basics. 11

4.2 Creating a Container Image app. 13

4.3 Diagnose. 16

4.4 Possible errors. 17

4.5 Snap templates. 18

5 ctrlX UI – Elements 27

5.1 Windows. 27

5.1.1 Window – “Images”. 27

5.1.2 Window – “Containers”. 27

6 Related documentation 29

6.1 Overview. 29

6.2 ctrlX AUTOMATION. 29

6.3 ctrlX WORKS. 29

6.4 ctrlX CORE. 30

6.5 ctrlX CORE Apps. 30

7 Service and support 34

8 Glossary 35

8.1 Docker. 35

8.2 Snapcraft. 35

9 Index 37

1 About this documentation

Editions of this documentation

Edition	Release date	Note
01	2022-12	First edition Container Engine App Version DOE-V-0116

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

4.1 Container Engine app - Basics

Container Engine app for ctrlX CORE

The ctrlX Container Engine app allows to run a Docker container on ctrlX CORE.

A Docker engine is provided in the ctrlX Container Engine app. The Docker REST API can be accessed via the ctrlX CORE reverse proxy.

A Docker image is deployed to the ctrlX CORE via a separate ctrlX Container Image app. In addition to the Docker image, this app contains the corresponding Docker configuration files required to run a Docker container application.

The most important terms are explained in the ➦ Glossary.

Introduction

The ctrlX Container Engine app provides a Docker engine on ctrlX CORE. It allows Docker images to be executed via a Docker container configuration file **docker-compose.yml**.

One or more Docker images can be installed on ctrlX CORE via separate Docker image apps. The images are configured and started via **docker-compose.yml**.

A snap template can be used to create a Docker image application for the ctrlX CORE from a Docker image and the corresponding Docker configuration files. This Container Image app, like all other ctrlX CORE apps, can be installed on a ctrlX CORE via the ctrlX Online Store or via the ctrlX Device Portal.

Installation and integration into the ctrlX CORE web interface

The app installation is described in the documentation at ctrlX CORE runtime, refer to the ➦ Web documentation.

By installing the app, the ctrlX CORE web interface around the “Container Engine” node with the windows “Images” and Containers in the ctrlX CORE web interface is extended, see

➦ Images

➦ Containers

Licensing

Required license to operate the Container Engine app on a ctrlX CORE control:

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

Users and authorizations

The Container Engine user authentication is connected to the ctrlX user management. To access the Container Engine app, authentication (login) is required in the ctrlX CORE control system. In the section “User & permissions”, Container engine-specific permissions can be specified, which control the data access to the Data Layer of the controller, see ➦ web documentation.

The following permissions exist for Container Engine:

- Manage Container Engine configuration
- Start/Stop Container and view Container Engine configuration
- View Container Engine configuration

In case of insufficient authorizations, sporadically no data is displayed or buttons are inactive.

Interfaces

The ctrlX Container Engine app includes a Docker engine with Docker daemon (dockerd) and Docker-Command-Line-Interface (Docker CLI). The app provides two content interfaces. Via these interfaces, data can be exchanged between the ctrlX Container Engine app and a ctrlX Container image app.

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

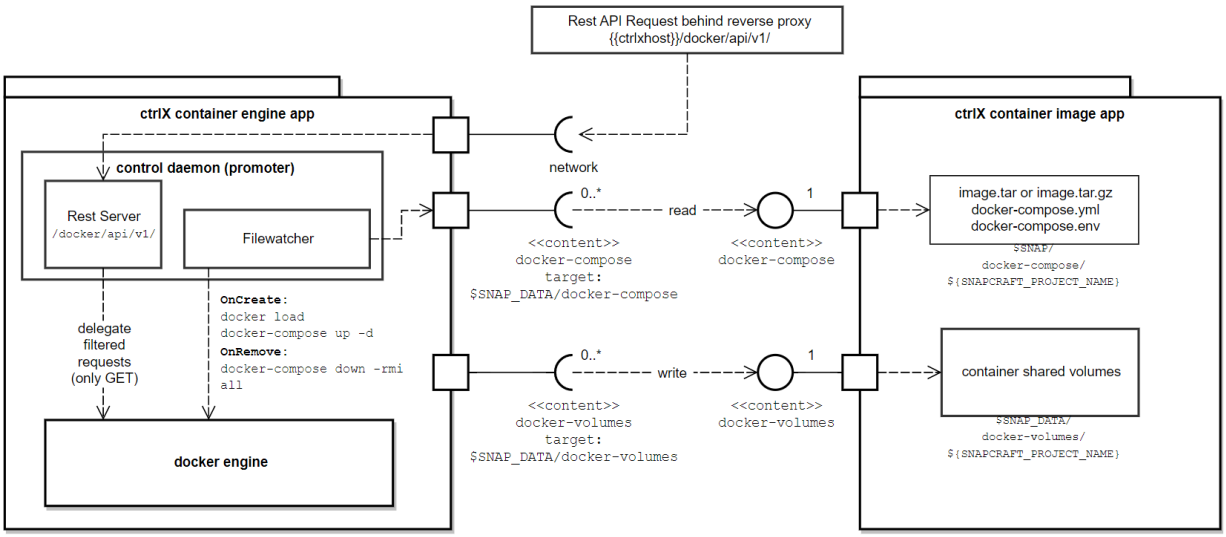

Fig. 1: Interfaces and data flow

Content interface "docker-compose

Using the Docker Compose interface, the Docker images and associated configuration files are provided to the ctrlX 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
```

Content interface "docker-volumes

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

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

of the Container Image app.

Here is an example excerpt from a docker-compose.yml for mapping a Docker volume with write access:

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

Docker Engine API

The REST interface of the Docker Engine API can be accessed via the https://{{host}}/docker/api/v1 address.

A full description of the Docker REST API can be found here:

➦ 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}}"
```

4.2 Creating a Container Image app

Creating a Container Image app

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

Creation and delivery/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. Please make sure that when using and distributing these images, the license terms are complied with, in particular for free and open source software (FOSS)

Prerequisites

To create a ctrlX Container Image app, you require the appropriate tools and knowledge of Docker and Snapcraft.

The Linux Ubuntu 20.04 LTS operating system is recommended as development platform.

For development purposes, Ubuntu 20.04 LTS can be run on a virtual machine. However, the virtualization should allow the use of snapd, for example, to download Snapcraft or Docker with the latest releases via the Snapcraft store. Snapd cannot be run in the current version of the Windows Subsystem for Linux (WSL).

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

It is recommended to use a new release of the Docker engine that supports the creation of multi-platform images. The latest Docker engine can e.g. be installed via the Snapcraft store (https://snapcraft.io/docker):

sudo snap install docker

To run Docker with the permissions of a standard user, this command sequence has to be entered:

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

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

Snapcraft

Snapcraft can be installed following these instructions: ➦ https://snapcraft.io/install/snapcraft/ubuntu

To use the latest release of Snapcraft on Ubuntu 20.04, Snapcraft should be installed via the Snapcraft Store (https://snapcraft.io/install/snapcraft/ubuntu):

sudo snap install snapcraft --classic

Creating a Docker image

First, create a Docker image and load it into the Docker Engine.

A Docker image can be created using the Docker Command Line Interface (Docker CLI). The relevant Docker documentation can be used for this purpose:

➦ https://docs.docker.com/reference/

An existing Docker image can be downloaded via ➦ Docker Hub. Here, as an example, via a shell script, the Docker image "eclipse-mosquitto" for the target platform arm64 is saved to a Docker image file via Docker CLI commands:

```
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}
```

Snap configuration

In the Snap template, the necessary interfaces are already set in snapcraft.yaml. Only a few positions need to be adjusted.

Here, the snapcraft configuration snapcraft.yaml of the mosquitto image app is shown as an example:

```
name: eclipse-mosquitto
version: '1.0'
base: core20
summary: eclipse-mosquitto docker image app
description: |
  This snap contains the docker image eclipse-mosquitto.
  The files image.tar, docker-compose.yml and docker-
compose.env files
  are provided via content-interface 'docker-compose'
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}
```


Docker Compose variables

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

Example of a docker-compose.env:

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

Find the documentation about Docker Compose variables under:

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

Docker Compose configuration

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

A tutorial for the Docker Compose configuration can be found here:

➦ https://docs.docker.com/compose/

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

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

```
      - ${SNAP_DATA}/docker-volumes/mosquitto-docker/log:/
mosquitto/log
    restart: on-failure
```

Here, Docker volumes with write access are mapped to a directory within the image app.

The directories

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

are accessible from the Docker container via the mapping in the Docker Compose configuration with write permissions.

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

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 the corresponding target architecture:

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

or

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

4.3 Diagnose

SSH console

Currently, the container engine and Container Image apps can only be diagnosed via the ctrlX UI to a limited extent. Therefore, the use of an SSH console with root privileges is recommended for comprehensive diagnostics (see ➦ Docker Command Line Interface on page 17).

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

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

Example:

```
ssh boschrexroth@192.168.1.1
```

Logging

The logs of the ctrlX Docker snap can be used to verify the loading and launching of the images. In an SSH console, the logs can be output with this command:

```
sudo snap logs ctrlx-docker –f
```

If SSH access is not possible, the logs can be viewed via the logbook in the ctrlX UI.

First, this setting has to be selected in the ctrlX UI:

Diagnostics | Logbook | Settings | Show system messages: true

Using the filter function on the services in the ctrlX Docker app, the corresponding logs can be displayed:

Diagnostics | Logbook | Filter | Units | snap.ctrx-docker.dockerd.service: true
Diagnostics | Logbook | Filter | Units | snap.ctrx-docker.dockerd.service: true

Docker Command Line Interface

The Docker CLI can be used to retrieve information from the Docker Engine and execute commands. The documentation can be found here:

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

Here are a few examples of important commands for diagnosing Docker images:

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

4.4 Possible errors

In this section, several potential error sources are pointed out when creating a Container Image app.

IP forwarding

If an http server is to be accessed in the Docker container, IP forwarding has to be set in the network settings. Select the setting in the ctrlX UI:

Settings | Connectivity | eth0 | Configuration: Enable IP forwarding: true

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.

docker pull eclipse-mosquitto:latest --platform amd64

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

Target architecture arm64, e.g. for ctrlX CORE M4:

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

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

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

Storage space

When selecting a Docker image in Docker Hub, take the image size into consideration. The memory space on the ctrlX CORE is limited. For example, on the ctrlX CORE M4, the Docker image should not exceed 300 MB in size. For Docker images based on a Linux image, it is always recommended to select the low weight Linux base image Alpine.

Reboot

The running containers in the Docker Engine should be restarted automatically in case of a ctrlX CORE reboot. In the configuration file docker-compose.yml this can be done via the option

restart: always

, see also ➦ https://docs.docker.com/compose/compose-file/compose-file-v3/#restart.

4.5 Snap templates

To create a Container Image app, one of the snap templates described here can be used. The templates contain all the necessary files to build a ctrlX Container Image app on in a build environment (see ➦ Prerequisites on page 13) .

Snap template shell scripts

Creating a ctrlX container image snap is done using shell scripts in the following steps.

Build Docker Content

The shell script build_content.sh is used to add or complete the contents of the docker-compose directory:

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

Build Snap

Using the build_snap.sh shell script, the snap build is first prepared and then created for the appropriate target platform:

1. Delete existing snap
2. Delete existing Snapcraft configuration
3. Rebuild snap for specific target architecture

Build All

With the help of the shell script build_all.sh the steps ➦ Build Docker Content and ➦ Build-Snap executed in the correct order and for the appropriate target platform.

Template “hello-web”

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

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
```

Package Assets

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

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": []
      }
    ]
  }
}
```

Dockerfiles:

Dockerfile amd64

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

Dockerfile 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
```

Snapcraft:

snapcraft.yaml

```
name: docker-hello-web
version: '1.18.0'
base: core20
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}
```

Docker compose:

docker-compose.yml

```
version: '3.7'
services:
  brc-web:
    container_name: "hello-web"  #unique container_name
    image: ${IMAGE_NAME}:${IMAGE_TAG}
    ports:
      - "8188:8188"
    stdin_open: false
    tty: false
    restart: on-failure
```

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 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}
```

Template "eclipse-mosquitto”

The template "eclipse-mosquitto" 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
```

Snapcraft

snapcraft.yaml

```
name: docker-mosquitto
version: '1.18.0'
base: core20
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}
```

Docker Compose

config

The Config directory is provided including the mosquitto.conf file with the Docker-Compose content interface. Thus, access in the Docker container to the mosquitto.conf file is possible via the volume mapping 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
```

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}
```

5 ctrlX UI – Elements

5.1 Windows

5.1.1 Window – “Images”

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

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

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

Call:

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

Elements of the “Images” window

GUI element	Description
Table	“Name” Image name
	“Tags” Image tags See ➦ link to web documentation
	“Maintainer” Image editor
	“Size” Image size
	“Created” Time of the creation of the image
	“Details” ⓘ By clicking on the button, the “Further information” window of the image opens

Further information

- ➦ Chapter 4.1 Container Engine app - Basics on page 11
- ➦ Chapter 5.1.2 Window – “Containers” on page 27

5.1.2 Window – “Containers”

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

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

Call:

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

Elements of the “Containers” **window**

GUI element	Description
Table	“Name” Container name
	“Version” Container version
	“Created” Time of creation of the container
	“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 and “Further information” is displayed
	“State” Container status
	“Actions” Contains more buttons ▷ “Start” Start container □ “Stop” Stop container ↻ “Restart” Container restart ⏸ “Pause” Pause container ✕ “Kill” Delete container
	“Details” ⓘ Click on the button to open a window with more information about the container with the “Diagnosis log” and “Further information” tabs. In the “Diagnosis log” tab, use ↻ to refresh the view

Further information

- ➦Chapter 4.1 Container Engine app - Basics on page 11
- ➦Chapter 5.1.1 Window – “Images” on page 27

6 Related documentation

6.1 Overview

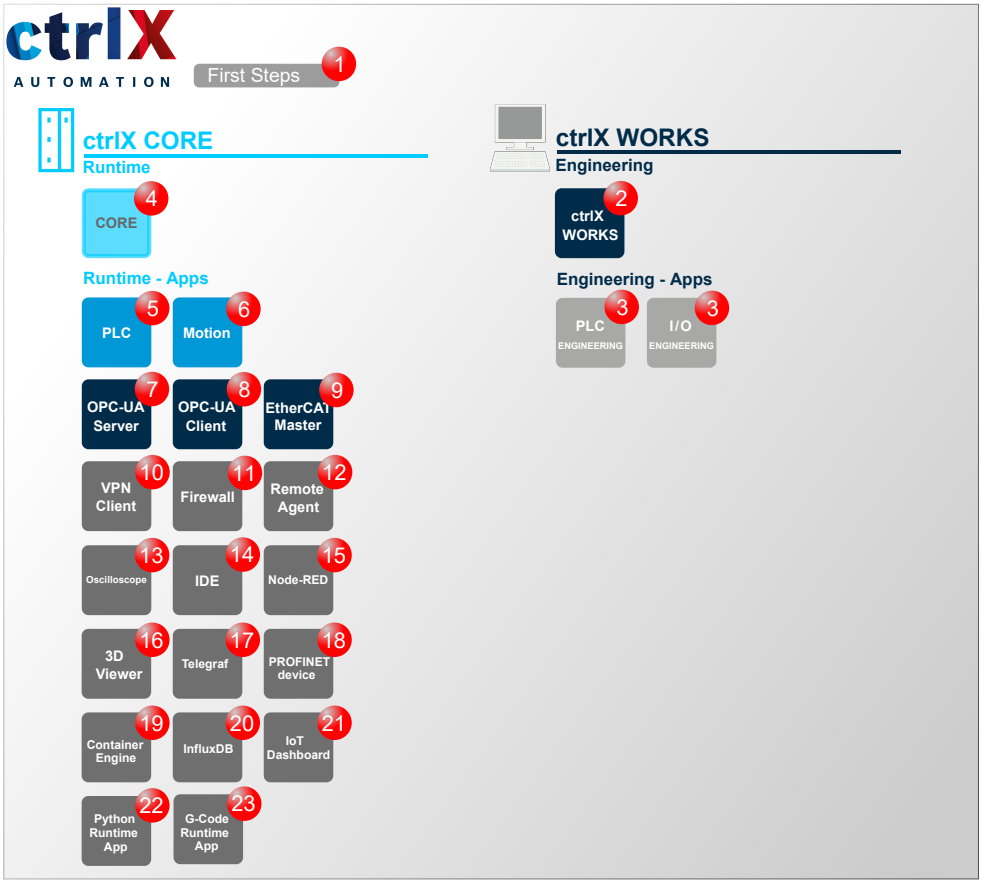

Fig. 2: Overview on further documentations

6.2 ctrlX AUTOMATION

No.	Documentation
1	**ctrlX WORKS First Steps 01VRS** Quick Start Guide ➦ Web documentation link Ordering information: • DOK-XWORKS-F*STEP**V01-QURS-EN-P • R911403760

6.3 ctrlX WORKS

No.	Documentation
2	**ctrlX WORKS Basic System 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XWORKS-********V01-APRS-EN-P • R911403761
3	**ctrlX PLC Engineering - PLC Programming System 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XPLC**-ENG*****V01-APRS-EN-P • R911403764
3	**ctrlX PLC Engineering - PLC Libraries 01VRS** Reference ➦ Web documentation link Ordering information: • DOK-XPLC**-LIBRARY*V01-RERS-EN-P • R911403766

6.4 ctrlX CORE

Nr.	Dokumentation
4	**ctrlX CORE - Runtime 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-BASE****V01-APRS-EN-P • R911403768
	ctrlX CORE - Nodes of the Data Layer 01VRS Reference ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-BASE*DL*V01-RERS-EN-P • R911420072
	ctrlX CORE - Diagnostics 01VRS Reference ➦ Web documentation link Ordering information: • DOK-XCORE*-DIAG****V01-RERS-EN-P • R911403770

6.5 ctrlX CORE Apps

Nr.	Dokumentation
5	**PLC App - PLC Runtime Environment for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-PLC*****V01-APRS-EN-P • R911403787
6	**Motion App - Motion Runtime Environment for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-MOTION**V01-APRS-EN-P • R911403791
7	**OPC UA Server App - OPC UA Server for ctrlX CORE 01VRS** Application Manual ➦ Link zur Web-Dokumentation Ordering information: • DOK-XCORE*-OPCSERV*V01-APRS-EN-P • R911403778
8	**OPC UA Client App - OPC UA Client for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-OPCCLIENV01-APRS-EN-P • R911403781
9	**EtherCAT Master App - EtherCAT Master for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-ETHERCATV01-APRS-EN-P • R911403773
10	**VPN Client App - Remote Support Software for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-VPN*****V01-APRS-EN-P • R911403775
11	**Firewall App - Security Functions for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-FIREWALLV01-APRS-EN-P • R911403783

Nr.	Dokumentation
12	**Remote Agent App - ctrlX Device Portal Connection for ctrlX Devices 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-REMOTE**V01-APRS-EN-P • R911403785
13	**Oscilloscope App - Oscilloscope Function for ctrlX Devices 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-OSCI****V01-APRS-EN-P • R911409806
14	**IDE App - Integrated Development Environment 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-IDE*****V01-APRS-EN-P • R911410625
15	**Node RED App - Graphic Programming for ctrlX CORE 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-NODERED*V01-APRS-EN-P • R911403789
16	**3D Viewer App - Browser-based 3D Kinematic Simulation for ctrlX CORE 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-3D*VIEW*V01-APRS-EN-P • R911416124
17	**Telegraf App - Server Agent for Collecting Data in the Data Layer 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-TELEGRAFV01-APRS-EN-P • R911416836
18	**PROFINET Device App - PROFINET Device for ctrlX CORE 01VRS** Application Manual ➥ Web documentation link Ordering information: • DOK-XCORE*-PROFINETV01-APRS-EN-P • R911417857

Nr.	Dokumentation
19	**Container Engine App - Use of Docker® Images on ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-DOCKER**V01-APRS-EN-P • R911417855
20	**InfluxDB App - Influx Database Connection for ctrlX CORE 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-INFLUX**V01-APRS-EN-P • R911418738
21	**IoT Dashboard App - Data Visualization in Dynamic, Interactive Dashboards 01VRS** Application Manual ➦ Web documentation link Ordering information: • DOK-XCORE*-GDB*****V01-APRS-EN-P • R911420426
22	**Python Runtime App - Python Runtime Environment for ctrlX CORE 01VRS** Application Manual ➦ Link zur Web-Dokumentation Ordering information: • DOK-XCORE*-PYR*****V01-APRS-EN-P • R911420430
23	**G-Code Runtime App - G-Code Interpreter for ctrlX CORE 01VRS** Application Manual ➦ Link zur Web-Dokumentation Ordering information: • DOK-XCORE*-GCO*****V01-APRS-EN-P • R911420428

7 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)

8 Glossary

8.1 Docker

A comprehensive and detailed glossary on Docker can be found here:
➦Web documentation link

Image

A Docker image is a core image of a container. An image consists of several layers that are read-only and cannot be modified. Multiple containers can be launched 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 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 web documentation

Engine

The Docker engine acts as a client-server application and includes:

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

See ➦link to web documentation

Compose

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

8.2 Snapcraft

A comprehensive and detailed glossary on Snap or Snapcraft can be found here:
➦Link to the web documentation

Snapcraft

Snapcraft is a command line tool for creating snaps and allows you to create snaps and thus apps for the ctrlX CORE. See ➦link to web documentation

Content interface

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

9 Index

C
Container Engine app
Glossary. 35
Container Engine App
Basics. 11
Creating a Container Image app. 13
ctrlX AUTOMATION
Related documentation. 29
D
Diagnose. 16
H
Helpdesk. 34
Hotline. 34
I
Intended use
Areas of application. 7
Areas of use. 7
Introduction. 7
P
Possible errors. 17
S
Safety instructions. 9
Service hotline. 34
Snap templates. 18
Support. 34
U
Unintended use. 8
Consequences, disclaimer. 7
W
Window
Containers. 27
Images. 27

rexroth
A Bosch Company

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

R911417855