rexroth
A Bosch Company

Anwendungsbeschreibung

Container Engine App

Verwendung von Docker® Images auf der ctrlX CORE 01VRS

R911417856, Ausgabe 01

Schutzvermerk

Verbindlichkeit

Die angegebenen Daten dienen allein der Produktbeschreibung und sind nicht als zugesicherte Eigenschaften im Rechtssinne zu verstehen. Änderungen im Inhalt der Dokumentation und Liefermöglichkeiten der Produkte sind vorbehalten.

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

c2e7b15c673acc930a347e885d054cbf, 1, de_DE

Inhaltsverzeichnis

1 Über diese Dokumentation

Ausgaben dieser Dokumentation

Ausgabe	Stand	Bemerkung
01	2022-12	Erstausgabe Container Engine App Version DOE-V-0116

2 Wichtige Gebrauchshinweise

2.1 Bestimmungsgemäßer Gebrauch

2.1.1 Einführung

Produkte von Rexroth werden nach dem jeweiligen Stand der Technik entwickelt und gefertigt.

Vor ihrer Auslieferung werden die Produkte auf ihren betriebssicheren Zustand hin überprüft.

⚠ WARNUNG	**Personen- und Sachschäden durch falschen Gebrauch der Produkte!** Die Produkte dürfen nur bestimmungsgemäß eingesetzt werden. Wenn die Produkte nicht bestimmungsgemäß eingesetzt werden, dann können Situationen entstehen, die Sach- und Personenbeschädigung nach sich ziehen.

HINWEIS	**Schäden bei nicht bestimmungsgemäßem Gebrauch** Für Schäden bei nicht bestimmungsgemäßem Gebrauch der Produkte leistet Rexroth als Hersteller keinerlei Gewährleistung, Haftung oder Schadensersatz. Die Risiken bei nicht bestimmungsgemäßem Gebrauch der Produkte liegen allein beim Anwender. Bevor Sie die Produkte der Firma Rexroth einsetzen, müssen die folgenden Voraussetzungen erfüllt sein, um einen bestimmungsgemäßen Gebrauch der Produkte zu gewährleisten: – Jeder, der in irgendeiner Weise mit Rexroth Produkten umgeht, muss die entsprechenden Sicherheitsvorschriften und den bestimmungsgemäßen Gebrauch lesen und verstehen – Sofern es sich bei den Produkten um Hardware handelt, müssen die Produkte in ihrem Originalzustand belassen werden; d. h. es dürfen keine baulichen Veränderungen an den Produkten vorgenommen werden. Softwareprodukte dürfen nicht dekompiliert werden und ihre Quellcodes dürfen nicht verändert werden – Beschädigte oder fehlerhafte Produkte dürfen nicht eingebaut oder in Betrieb genommen werden – Es muss gewährleistet sein, dass die Produkte entsprechend den in der Dokumentation genannten Vorschriften installiert sind

2.1.2 Einsatz- und Anwendungsbereiche

Produkte der ctrlX Baureihe sind für Motion-/Logic-Anwendungen geeignet.

HINWEIS	Produkte der ctrlX Baureihe dürfen nur mit den in dieser Dokumentation angegebenen Zubehör- und Anbauteilen benutzt werden. Nicht ausdrücklich genannte Komponenten dürfen weder angebaut noch angeschlossen werden. Gleiches gilt für Kabel und Leitungen. Der Betrieb darf nur in den ausdrücklich angegebenen Konfigurationen und Kombinationen der Hardware-Komponenten und mit der in den jeweiligen Dokumentationen und den Funktionsbeschreibungen angegebenen und spezifizierten Soft- und Firmware erfolgen.

Produkte der ctrlX Baureihe sind für den Einsatz in ein- und mehrachsigen Antriebs- und Steuerungsaufgaben geeignet. Für den applikationsspezifischen Einsatz des Systems stehen Gerätetypen mit unterschiedlicher Ausstattung und unterschiedlichen Schnittstellen zur Verfügung.

Typische Anwendungsbereiche:

- Gebäudeautomatisierung
- IoT und Security Gateway bzw. Device
- Handling & Robotic

Steuerungen der ctrlX CORE Baureihe dürfen nur unter den in den weiterführenden Dokumentationen angegebenen Montage- und Installationsbedingungen, in der angegebenen Gebrauchslage und unter den angegebenen Umweltbedingungen (Temperatur, Schutzart, Feuchte, EMV u. a.) betrieben werden.

2.2 Nicht bestimmungsgemäßer Gebrauch

Die Verwendung von ctrlX-Produkten außerhalb der vorgenannten Anwendungsgebiete oder unter anderen als den in der Dokumentation beschriebenen Betriebsbedingungen und angegebenen technischen Daten gilt als "nicht bestimmungsgemäß".

ctrlX-Produkte dürfen nicht eingesetzt werden, wenn sie den folgenden Bedingungen ausgesetzt sind:

- Betriebsbedingungen, die die vorgeschriebenen Umgebungsbedingungen nicht erfüllen. Untersagt sind z. B. der Betrieb unter Wasser, unter extremen Temperaturschwankungen oder extremen Maximaltemperaturen
- Bei Anwendungen, die von Rexroth nicht ausdrücklich freigegeben sind

3 Sicherheitshinweise

Die Sicherheitshinweise, soweit in der vorliegenden Anwendungsdokumentation vorhanden, beinhalten bestimmte Signalwörter ("Gefahr", "Warnung", "Vorsicht", "Hinweis") und ggf. eine Signalgrafik (nach ANSI Z535.6-2006).

Das Signalwort soll die Aufmerksamkeit auf den Sicherheitshinweis lenken und bezeichnet die Schwere der Gefährdung.

Die Signalgrafik (Warndreieck mit Ausrufezeichen), welche den Signalwörtern "Gefahr", "Warnung" und "Vorsicht" vorangestellt wird, weist auf Gefährdungen für Personen hin.

Die Sicherheitshinweise in dieser Dokumentation werden wie folgt dargestellt:

⚠ GEFAHR	Bei Nichtbeachtung dieses Sicherheitshinweises **werden** Tod oder schwere Körperverletzung eintreten.
⚠ WARNUNG	Bei Nichtbeachtung dieses Sicherheitshinweises **können** Tod oder schwere Körperverletzung eintreten.
⚠ VORSICHT	Bei Nichtbeachtung dieses Sicherheitshinweises können mittelschwere oder leichte Körperverletzung eintreten.
HINWEIS	Bei Nichtbeachtung dieses Sicherheitshinweises können Sachschäden eintreten.

4 Einführung und Übersicht

4.1 Container Engine App – Grundlagen

Container Engine App für ctrlX CORE

Die ctrlX Container Engine App ermöglicht die Ausführung eines Docker-Containers auf der ctrlX CORE.

In der ctrlX Container Engine App wird eine Docker-Engine zur Verfügung stellt. Die Docker-REST-API kann über den Reverse-Proxy der ctrlX CORE angesprochen werden.

Ein Docker-Image wird über eine separate ctrlX Container Image App auf die ctrlX CORE ausgebracht. Diese App enthält neben dem Docker-Image die entsprechenden Docker-Konfigurationsdateien, die zur Ausführung einer Docker-Container-Anwendung notwendig sind.

Die wichtigsten Begriffe sind im ➦ Glossar erklärt.

Einführung

Die ctrlX Container Engine App stellt eine Docker-Engine auf einer ctrlX CORE zur Verfügung. Sie erlaubt die Ausführung von Docker-Images über eine Docker-Container-Konfigurationsdatei **docker-compose.yml**.

Ein oder mehrere Docker Images können über separate Docker-Image-Apps auf die ctrlX CORE installiert werden. Die Images werden über die **docker-compose.yml** konfiguriert und gestartet.

Mit Hilfe eines Snap-Templates kann aus einem Docker-Image und den entsprechenden Docker-Konfigurationsdateien eine Docker-Image-Anwendung für die ctrlX CORE erstellt werden. Diese Container-Image-App kann, wie alle anderen ctrlX CORE Apps, über den ctrlX Online Store bzw. über das ctrlX Device Portal auf einer ctrlX CORE installiert werden.

Installation und Einbindung in die ctrlX CORE Web-Oberfläche

Die Installation von Apps ist in der Dokumentation zur ctrlX CORE Runtime beschrieben, siehe ➦ Web-Dokumentation.

Durch die Installation der App wird die ctrlX CORE Web-Oberfläche um den „Container Engine“-Knoten mit den Fenstern „Images“ und Containers in der ctrlX CORE Web-Oberfläche erweitert, siehe

➦ Images
➦ Containers

Lizenzierung

Erforderliche Lizenz zum Betrieb der Container Engine App auf einer ctrlX CORE Steuerung:

Typschlüssel	Materialnummer
SWL-XC*-DOE-DOCKERENGINE*-NNNN	R911409943

Benutzer und Berechtigungen

Die Container Engine Benutzerauthentifizierung ist an die ctrlX-Benutzerverwaltung angebunden. Für den Zugriff auf die Container Engine App ist eine Authentifizierung (Anmeldung) in der ctrlX CORE Steuerung erforderlich. Im Abschnitt „Benutzer & Berechtigungen“ können Container Engine spezifische Berechtigungen festgelegt werden, die den Datenzugriff auf den Data Layer der Steuerung regeln, siehe ➦ Web-Dokumentation.

Folgende Berechtigungen gibt es für Container Engine:

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

Nicht ausreichende Berechtigungen können dazu führen, dass keine Daten angezeigt werden können bzw. Schaltflächen keine Funktion haben.

Schnittstellen

Die ctrlX Container Engine App beinhaltet eine Docker-Engine mit Docker-Daemon (dockerd) und dem Docker-Command-Line-Interface (Docker CLI). Die App stellt zwei Content-Interfaces zur Verfügung. Über diese Schnittstellen können Daten zwischen der ctrlX Container Engine App und einer ctrlX Container Image App ausgetauscht werden.

Das folgende UML-Diagramm stellt die Schnittstellen und den Datenfluss schematisch dar:

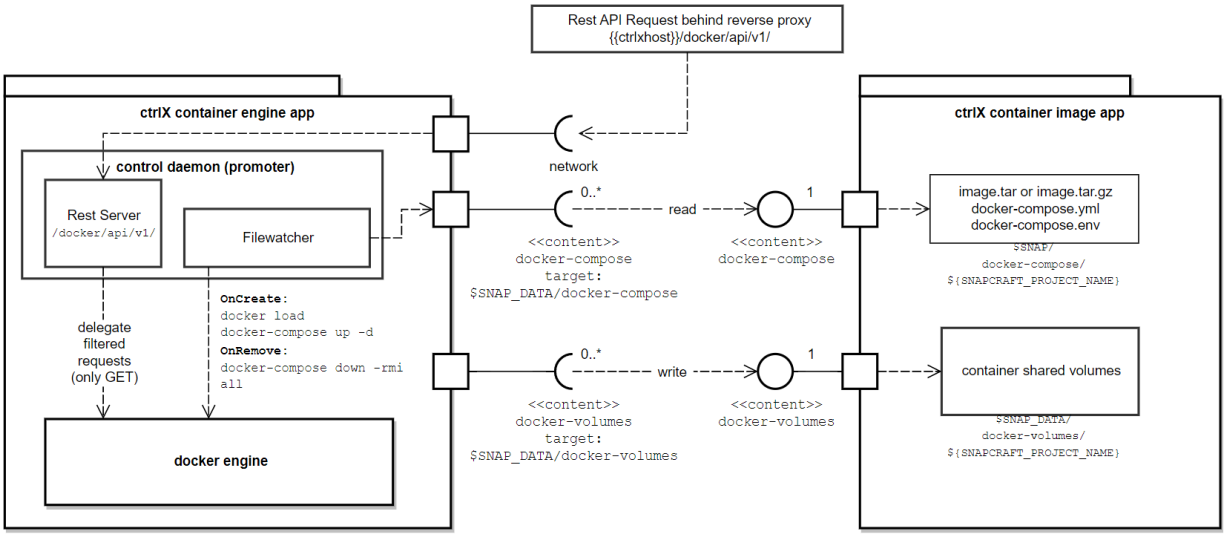

Abb. 1: Schnittstellen und Datenfluss

Content Interface "docker-compose"

Über die Docker-Compose-Schnittstelle werden die Docker-Images und die zugehörigen Konfigurationsdateien der ctrlX Container Engine App zur Verfügung gestellt. Folgende Dateien werden über Docker-Compose bereitgestellt:

```
└── 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"

Die Content-Schnittstelle Docker-Volumes erlaubt den Schreibzugriff von einem Docker-Container auf ein Verzeichnis der Image-App. Ein Docker-Container kann aus der ctrlX Container Engine App heraus über diese Schnittstelle auf ein schreibbares Verzeichnis

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

der Container-Image-App schreiben.

Hier ein exemplarischer Auszug aus einer docker-compose.yml für die Zuordnung eines Docker-Volumes mit Schreibzugriff:

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

Docker-Engine-API

Die REST-Schnittstelle der Docker-Engine-API kann über die Adresse https://{{host}}/docker/api/v1 angesprochen werden.

Eine vollständige Beschreibung der Docker-REST-API finden Sie hier:

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

Beispiel für ein REST Befehl via curl - hier das Abrufen der vorhandenen Docker-Images als json:

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

4.2 Erstellen einer Container-Image-App

Erstellen einer Container-Image-App

Bosch Rexroth stellt selbst keine Docker-Files und Docker-Images zur kommerziellen Nutzung zur Verfügung.

Erstellung und Auslieferung/Vertrieb von Docker-Images mit der ctrlX Container App liegt in der Verantwortung des Nutzers.

Die nachfolgende Anleitung beschreibt die technischen Schritte zur Erstellung eines Images. Bitte stellen Sie sicher, dass bei der Nutzung und Verbreitung dieser Images, die lizenzrechtlichen Bedingungen berücksichtigt sind, speziell bei freier und Open-Source-Software (FOSS)

Voraussetzungen

Für das Erstellen einer ctrlX Container Image App müssen die entsprechenden Tools und Kenntnisse zu Docker und Snapcraft vorhanden sein.

Als Entwicklungsplattform wird das Betriebssystem Linux Ubuntu 20.04 LTS empfohlen.

Für die Entwicklung kann Ubuntu 20.04 LTS in einer virtuellen Maschine betrieben werden. Allerdings sollte die Virtualisierung den Einsatz von snapd erlauben, um z. B. Snapcraft bzw. Docker mit den neuesten Releases über den Snapcraft-Store beziehen zu können. In der aktuellen Version von Windows-Subsystem für Linux (WSL) kann Snapd nicht betrieben werden.

Docker

Um eine Container-Image-App zu erstellen, muss ein Docker-Image (image.tar oder image.tar.gz) vorliegen.

Die Installation einer Docker-Engine auf Ubuntu wird hier beschrieben: ➦https://docs.docker.com/engine/install/ubuntu/

Es wird empfohlen, ein neues Release der Docker-Engine zu verwenden, dass die Erstellung von Multi-Platform-Images unterstützt. Die Installation z. B. der neuesten Docker-Engine kann z. B. über den Snapcraft-Store (https://snapcraft.io/docker) erfolgen:

sudo snap install docker

Um Docker mit den Berechtigungen eines Standard-Users ausführen zu können, muss diese Befehlsfolge eingegeben werden:

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

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

Snapcraft

Die Installation von Snapcraft kann nach dieser Anleitung durchgeführt werden: ➦https://snapcraft.io/install/snapcraft/ubuntu

Um das neueste Release von Snapcraft auf Ubuntu 20.04 verwenden zu können, sollte Snapcraft über den Snapcraft-Store (https://snapcraft.io/install/snapcraft/ubuntu) installiert werden:

sudo snap install snapcraft --classic

Erstellen eines Docker-Image

Zunächst muss ein Docker-Image angelegt werden und in die Docker-Engine geladen werden.

Ein Docker-Image kann über das Docker-Command-Line-Interface (Docker CLI) erzeugt werden. Hierzu können die entsprechenden Dokumentationen von Docker verwendet werden:

➦https://docs.docker.com/reference/

Ein bestehendes Docker-Image kann über ➦ Docker-Hub heruntergeladen werden. Hier wird exemplarisch, über ein Shell-Skript, das Docker-Image „eclipse-mosquitto“ für die Zielplattform arm64 über Docker-CLI-Befehle in eine Docker-Image-Datei gespeichert:

```
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-Konfiguration

Im Snap-Template sind die notwendigen Schnittstellen in der snapcraft.yaml bereits gesetzt. Es müssen nur wenige Stellen angepasst werden.

Hier wird exemplarisch die Snapcraft-Konfiguration snapcraft.yaml der mosquitto Image App gezeigt:

```
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-Variablen

Die Umgebungsvariablen können in der Datei docker-compose.env angelegt werden. Auf diese Variablen kann in der docker-compose.yml zugegriffen werden.

Beispiel einer docker-compose.env:

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

Hier finden Sie eine Dokumentation zum Thema Docker-Compose-Variablen:
➦ https://docs.docker.com/compose/environment-variables/

Docker-Compose-Konfiguration

Die Docker-Anwendung wird über die Datei docker-compose.yml konfiguriert.

Eine Anleitung für die Docker-Compose-Konfiguration finden Sie hier:

➦ https://docs.docker.com/compose/

Beispiel einer docker-compose.yml für 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
```

Hier werden die Docker-Volumes mit Schreibzugriff auf ein Verzeichnis innerhalb der Image-App zugeordnet.

Die Verzeichnisse

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

sind aus dem Docker-Container über die Zuordnung in der Docker-Compose-Konfiguration mit Schreibrechten erreichbar.

Die Option "restart: on-failure" sollte immer gesetzt sein, damit die Docker-Container auch im Fehlerfall gestartet werden können.

Image-App erstellen

Um den Snap mit dem Docker-Content zu erstellen, müssen in der Konsole folgende Snapcraft-Befehle ausgeführt werden:

Snapcraft-Build-Verzeichnisse aufräumen:

```
snapcraft clean
```

Snap mit entsprechender Zielarchitektur erzeugen:

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

oder

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

4.3 Diagnose

SSH-Konsole

Aktuell können die Container-Engine und Container-Image-Apps nur eingeschränkt über die ctrlX UI diagnostiziert werden. Daher ist für die umfassende Diagnose die Verwendung einer SSH-Konsole mit Root-Rechten empfohlen (siehe ➥ Docker-Command-Line-Interface auf Seite 17).

Eine SSH-Konsole kann unter Windows mit folgendem Befehl geöffnet werden:

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

Beispiel:

```
ssh boschrexroth@192.168.1.1
```

Logging

Mit den Logs des ctrlX Docker Snaps kann das Laden und Starten der Images überprüft werden. In einer SSH-Konsole können die Logs mit diesem Befehl ausgegeben werden:

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

Falls der SSH-Zugriff nicht möglich ist, können die Logs über das Logbuch in der ctrlX UI eingesehen werden.

Zunächst muss diese Einstellung in der ctrlX UI vorgenommen werden:

Diagnostics | Logbook | Settings | Show system messages: true

Dann kann über die Filterfunktion auf die Dienste in der ctrlX Docker App die entsprechenden Logs angezeigt werden:

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

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

Docker-Command-Line-Interface

Mit der Docker-CLI können Information aus der Docker-Engine abgerufen und Kommandos ausgeführt werden. Eine vollständige Dokumentation kann hier eingesehen werden:

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

Hier ein paar Beispiele von wichtigen Kommandos zur Diagnose von Docker-Images:

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

4.4 Mögliche Fehler

In diesem Abschnitt soll auf einige mögliche Fehlerquellen bei der Erstellung einer Container-Image-App hingewiesen werden.

IP-Forwarding

Falls auf ein Http-Server im Docker-Container zugegriffen werden soll, muss in den Netzwerkeinstellungen das IP-Forwarding gesetzt sein. Hier kann die Einstellung in der ctrlX UI vorgenommen werden:

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

Zielarchitektur

Beim Erstellen einer Docker-Image-Datei muss die richtige Architektur für das Zielsystem angegeben werden. Falls das Docker-Image über ein Docker-Repository wie Docker-Hub geladen wird, kann die Zielarchitektur mit dem Schalter "platform" übergeben werden.

docker pull eclipse-mosquitto:latest --platform amd64

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

Zielarchitektur arm64, z. B. für ctrlX CORE M4:

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

Zielarchitektur arm64, z. B. für ctrlX CORE Virtual:

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

Speicherplatz

Bei der Auswahl eines Docker-Images in Docker-Hub sollte auf die Größe des Images geachtet werden. Der Speicherplatz auf der ctrlX CORE ist begrenzt. Auf der ctrlX CORE M4 z. B. sollte das Docker-Image die Größe von 300 MB nicht überschreiten. Für Docker-Images, die auf ein Linux-Image basieren, empfiehlt es sich immer das leichtgewichtige Linux-Basis-Image Alpine auszuwählen.

Reboot

Die laufenden Container in der Docker-Engine sollten im Falle eines Reboots der ctrlX CORE wieder automatisch gestartet werden. In der Konfigurationsdatei docker-compose.yml kann dies über die Option

restart: always

eingestellt werden, siehe auch ➦ https://docs.docker.com/compose/compose-file/compose-file-v3/#restart.

4.5 Snap-Templates

Um eine Container-Image-App zu erstellen, kann eines der hier beschriebenen Snap-Templates verwendet werden. Die Templates enthalten alle notwendigen Dateien, um eine ctrlX Container Image App auf einer entsprechenden Build-Umgebung (siehe ➦ Voraussetzungen auf Seite 13) zu bauen.

Snap-Template Shell-Skripte

Das Erzeugen eines ctrlX Container Image Snaps wird mit Hilfe von Shell-Skripten in folgenden Schritten vorgenommen.

Build-Docker-Content

Mit Hilfe des Shell-Skripts build_content.sh wird der Inhalt des Verzeichnisses docker-compose ergänzt bzw. vervollständigt:

1. Docker-Image-Archive ‚image.tar' erstellen
2. Docker-Variablendatei ‚docker-compose.env' erstellen

Build Snap

Mit Hilfe des Shell-Skripts build_snap.sh wird zunächst der Snap-Build vorbereitet und anschließend für die entsprechende Zielplattform erstellt:

1. Bestehenden Snap löschen
2. Bestehende Snapcraft-Konfiguration löschen
3. Snap für bestimmte Zielarchitektur neu erstellen

Build All

Mit Hilfe des Shell-Skripts build_all.sh werden die Schritte ➦ Build-Docker-Content und ➦ Build-Snap in der richtigen Reihenfolge und für die entsprechende Zielplattform ausgeführt.

Template „hello-web"

Das Template „hello-web" demonstriert die Erstellung eines einfachen Docker-Images über ein Docker-File.

Dateistruktur

```
.
├── 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

Das Verzeichnis configs wird inklusive der Datei package-manifest.json mit dem Content-Interface Package-Assets bereitgestellt. Hier ist die Konfiguration der Package-Assets dokumentiert: ➦ 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“

Das Template „eclipse-mosquitto“ demonstriert die Verwendung eines bestehenden Docker-Images im Docker-Hub (➦ https://hub.docker.com/).

Dateistruktur

```
.
├── 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

Das Verzeichnis Config wird inklusive der Datei mosquitto.conf mit dem Content Interface Docker-Compose bereitgestellt. Somit ist ein Zugriff im Docker-Container auf die Datei mosquitto.conf über die Volume-Zuordnung in der docker-compose.yml möglich.

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 Skripte

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 Bedienoberfläche – Elemente

5.1 Fenster

5.1.1 Fenster – „Images“

Das Fenster „Images“ zeigt die Liste der in Container Engine installierten Docker-Images an.

Zu jedem Image werden einige Eigenschaften, wie z. B. Name und Größe angezeigt. Über „Details“ können alle Eigenschaften eines Images angezeigt werden.

Verwandte Themen:
➦ Informationen zu Container Engine und zur Container Engine App

Aufruf:

ctrlX CORE Seitennavigation *„Container Engine → Images“*

Elemente des Fensters „Images“

Oberflächenelement	Beschreibung
Tabelle	„Name“ Name des Images
	„Tags“ Tags des Images Siehe ➦ Link zur Web-Dokumentation
	„Maintainer“ Herausgeber des Images
	„Size“ Größe des Images
	„Erstellt“ Zeitpunkt der Erstellung des Images
	„Details“ ⓘ Beim Klick auf die Schaltfläche öffnet sich das Fenster „Weitere Informationen“ zum Image

Weiterführende Informationen

- ➦ Kapitel 4.1 Container Engine App – Grundlagen auf Seite 11
- ➦ Kapitel 5.1.2 Fenster – „Containers“ auf Seite 27

5.1.2 Fenster – „Containers“

Das Fenster „Containers“ zeigt die Liste der Container in der Container Engine und deren aktuellen Zustand an

Verwandte Themen:
➦ Informationen zu Container Engine und zur Container Engine App

Aufruf:

ctrlX CORE Seitennavigation *„Container Engine → Containers“*

Elemente des Fensters „Containers“

Oberflächenelement	Beschreibung
Tabelle	„Name“ Name des Containers
	„Version“ Version des Containers
	„Erstellt“ Zeitpunkt der Erstellung des Containers
	„IP-Adresse“ IP-Adresse des Containers
	„Ports“ Ports des Containers
	„Image“ Das Image wird angezeigt Beim Klick auf den Image-Namen gelangen Sie in das Fenster „Images“ und Ihnen werden „Weitere Informationen“ angezeigt
	„State“ Status des Containers
	„Aktionen“ Enthält weitere Schaltflächen ▷ „Start“ Container starten □ „Stopp“ Container stoppen ↻ „Restart“ Neustart des Containers ⏸ „Pause“ Container pausieren ✕ „Kill“ Container löschen
	„Details“ ⓘ Beim Klick auf die Schaltfläche öffnet sich ein Fenster mit weiteren Informationen zum Container mit den Registerkarten „Log-Meldungen“ und „Weitere Informationen“. In der Registerkarte „Log-Meldungen“ können Sie über ↻ die Ansicht aktualisieren

Weiterführende Informationen

- ➦Kapitel 4.1 Container Engine App – Grundlagen auf Seite 11
- ➦Kapitel 5.1.1 Fenster – „Images“ auf Seite 27

6 Weiterführende Dokumentationen

6.1 Übersicht

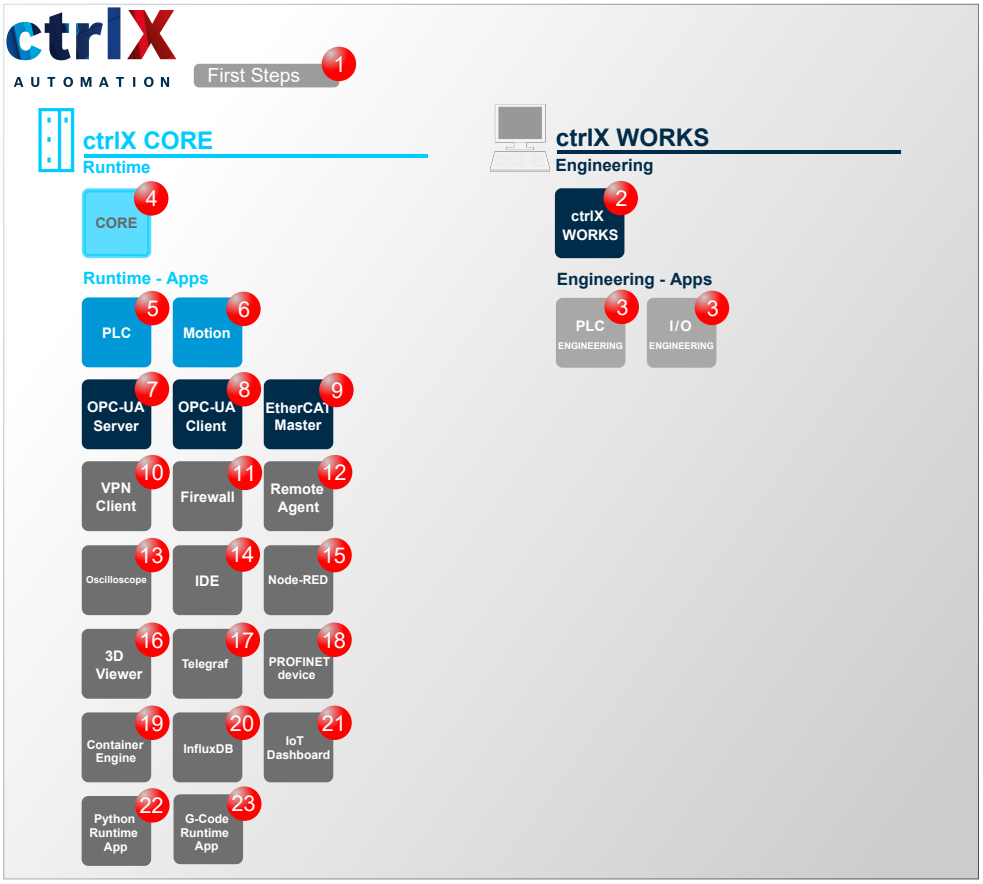

Abb. 2: Übersicht der weiterführenden Dokumentationen

6.2 ctrlX AUTOMATION

Nr.	Dokumentation
1	**ctrlX WORKS - Erste Schritte 01VRS** Quick Start Guide ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XWORKS-F*STEP**V01-QURS-DE-P • R911403759

6.3 ctrlX WORKS

Nr.	Dokumentation
2	**ctrlX WORKS - Basissystem 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XWORKS-********V01-APRS-DE-P • R911403762
3	**ctrlX PLC Engineering - SPS-Programmiersystem 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XPLC**-ENG*****V01-APRS-DE-P • R911403763
3	**ctrlX PLC Engineering - SPS-Bibliotheken 01VRS** Referenz ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XPLC**-LIBRARY*V01-RERS-DE-P • R911403765

6.4 ctrlX CORE

Nr.	Dokumentation
4	**ctrlX CORE - Runtime 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-BASE****V01-APRS-DE-P • R911403767
	ctrlX CORE - Knoten des Data Layer 01VRS Referenz ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-BASE*DL*V01-RERS-DE-P • R911420071
	ctrlX CORE - Diagnosen 01VRS Referenz ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-DIAG****V01-RERS-DE-P • R911403769

6.5 ctrlX CORE Apps

Nr.	Dokumentation
5	**PLC App - SPS-Laufzeitumgebung für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-PLC*****V01-APRS-DE-P • R911403786
6	**Motion App - Motion-Laufzeitumgebung für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-MOTION**V01-APRS-DE-P • R911403790
7	**OPC UA Server App - OPC UA Server für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-OPCSERV*V01-APRS-DE-P • R911403776
8	**OPC UA Client App - OPC UA Client für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-OPCCLIENV01-APRS-DE-P • R911403779
9	**EtherCAT Master App - EtherCAT Master für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-ETHERCATV01-APRS-DE-P • R911403771
10	**VPN Client App - Fernwartungssoftware für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-VPN*****V01-APRS-DE-P • R911403774
11	**Firewall App - Security Funktionen für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-FIREWALLV01-APRS-DE-P • R911403782

Nr.	Dokumentation
12	**Remote Agent App - ctrlX Device Portal-Anbindung für ctrlX-Geräte 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-REMOTE**V01-APRS-DE-P • R911403784
13	**Oscilloscope App - Oszilloskopfunktion für ctrlX-Geräte 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-OSCI****V01-APRS-DE-P • R911409805
14	**IDE App - Integrated Development Environment 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-IDE*****V01-APRS-DE-P • R911410624
15	**Node-RED App - Grafische Programmierung für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-NODERED*V01-APRS-DE-P • R911403788
16	**3D Viewer App - Browserbasierte 3D-Kinematik-Simulation für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-3D*VIEW*V01-APRS-DE-P • R911416123
17	**Telegraf App - Server-Agent zum Sammeln von Daten im Data Layer 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-TELEGRAFV01-APRS-DE-P • R911416837
18	**PROFINET Device App - PROFINET device für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-PROFINETV01-APRS-DE-P • R911417858

Nr.	Dokumentation
19	**Container Engine App - Verwendung von Docker® Images auf der ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-DOCKER**V01-APRS-DE-P • R911417856
20	**InfluxDB App - Influx-Datenbankanbindung für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-INFLUXD*V01-APRS-DE-P • R911418737
21	**IoT Dashboard App - Datenvisualisierung in dynamischen, interaktiven Dashboards 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-GDB*****V01-APRS-DE-P • R911420427
22	**Python Runtime App - Python-Laufzeitumgebung für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-PYR*****V01-APRS-DE-P • R911420431
23	**G-Code Runtime App - G-Code Interpreter für ctrlX CORE 01VRS** Anwendungsbeschreibung ➦ Link zur Web-Dokumentation Bestellinformationen: • DOK-XCORE*-GCO*****V01-APRS-DE-P • R911420429

7 Service und Support

Für Ihre schnelle und optimale Unterstützung verfügen wir über ein dichtes weltweites Servicenetz. Unsere Experten stehen Ihnen mit Rat und Tat zur Seite. Sie erreichen uns täglich **rund um die Uhr – auch an Wochenenden und Feiertagen**.

Service Deutschland

Unser technologieorientiertes Competence Center in Lohr deckt alle Belange rund um den Service für elektrische Antriebe und Steuerungen ab.

Sie erreichen unsere **Service-Hotline** und unseren **Service-Helpdesk** unter:

Telefon:	**+49 9352 40 5060**
Fax:	**+49 9352 18 4941**
E-Mail:	➦ service.svc@boschrexroth.de
Internet:	➦ http://www.boschrexroth.com

Auf unseren Internetseiten finden Sie ergänzende Hinweise zu Service, Reparatur (z. B. Anlieferadressen) und Training.

Service weltweit

Außerhalb Deutschlands nehmen Sie bitte zuerst Kontakt mit Ihrem Ansprechpartner auf. Die Hotline-Rufnummern entnehmen Sie bitte den Vertriebsadressen im Internet.

Vorbereitung der Informationen

Wir können Ihnen schnell und effizient helfen, wenn Sie folgende Informationen bereithalten:

- Eine detaillierte Beschreibung der Störung und der Umstände
- Angaben auf dem Typenschild der betreffenden Produkte, insbesondere Typenschlüssel und Seriennummern
- Ihre Kontaktdaten (Telefon-, Faxnummer und E-Mail-Adresse)

8 Glossar

8.1 Docker

Ein umfassendes und detailliertes Glossar zu Docker findet Sie hier:
➦ Link zur Web-Dokumentation

Image

Ein Docker-Image ist ein Speicherabbild eines Containers. Ein Image besteht aus mehreren Ebenen (Layer), die schreibgeschützt und nicht verändert werden können. Aus einem Image können mehrere Container gestartet werden.

Container

Als Container wird die Laufzeitinstanz eines Images bezeichnet.

Ein Docker-Container besteht aus:

- Docker-Image
- Ausführungsumgebung
- Satz von Anweisungen

Siehe ➦ Link zur Web-Dokumentation

Volumes

Docker-Volumes stellen auf dem Host-System ein Dateisystem mit oder ohne Schreibzugriff zur Verfügung, das vom Container verwendet werden kann. Siehe ➦ Link zur Web-Dokumentation

Engine

Die Docker-Engine fungiert als Client-Server-Anwendung und beinhaltet:

- Ein Server mit einem Daemon-Prozess dockerd
- Schnittstellen, über die Programme mit dem Docker-Daemon kommunizieren und ihn anweisen können
- Eine Befehlszeilenschnittstelle (CLI)

Siehe ➦ Link zur Web-Dokumentation

Compose

Compose ist ein Tool zur Konfiguration und Ausführung komplexer Anwendungen mit Docker. Mit Compose wird eine Multi-Container-Anwendung in einer einzigen Datei definiert. Diese Anwendung kann mit Hilfe der Compose-Datei (docker-compose.yml) mit einem einzigen Befehl gestartet werden. Siehe ➦ Link zur Web-Dokumentation

8.2 Snapcraft

Ein umfassendes und detailliertes Glossar zu Snap bzw. Snapcraft finden Sie hier:
➦ Link zur Web-Dokumentation

Snapcraft

Snapcraft ist ein Befehlszeilentool zum Erstellen von Snaps und ermöglicht die Erstellung von Snaps und somit Apps für die ctrlX CORE. Siehe ➦ Link zur Web-Dokumentation

Content Interface

Das Snapcraft-Content-Interface ermöglicht die gemeinsame Nutzung von Code und Daten von einem Producer-Snap zu einem Consumer-Snap. Siehe ➦ Link zur Web-Dokumentation

9 Index

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

R911417856